Camellia: 様々な環境に適した 128 ビットブロック暗号

青木和麻呂 † 市川哲也 ‡ 神田雅透 †
松井充 ‡ 盛合志帆 † 中嶋純子 ‡ 時田俊雄 ‡

† 日本電信電話株式会社
〒 239-0847 神奈川県横須賀市光の丘 1-1
{maro,kanda,shiho}@isl.ntt.co.jp

‡ 三菱電機株式会社
〒 247-8501 神奈川県鎌倉市大船 5-1-1
{ichikawa,matsui,june15,tokita}@iss.isl.melco.co.jp

第 1.0 版: 2000 年 7 月 13 日
(翻訳: 2000 年 9 月 21 日)
第 1.1 版: 2001 年 8 月 28 日
第 2.0 版: 2001 年 9 月 26 日

要旨 本稿では、新しい 128 ビットブロック暗号 *Camellia* を提案する。 Camellia は、ブロック長が 128 ビット、鍵長が 128/192/256 ビットのいずれかを利用でき、次期米国政府標準暗号 (AES; Advanced Encryption Standard) のインタフェースに準拠している。Camellia の特徴は、非常に高い安全性に加えて、ソフトウェアとハードウェアの両面で効率的な実装が可能な点である。まず安全性では Camellia は差分攻撃、線形攻撃に対して十分に安全であることを確認した。また、効率性ではソフトウェア、ハードウェアを問わず、 AES 最終候補暗号 (MARS, RC6, Rijndael, Serpent, Twofish) と比較して同程度以上の暗号化速度を実現している。アセンブリ言語による最適化された実装では、Pentium III (1.13GHz) 上で 471 Mbps の暗号化速度を実現できる。

さらに、特筆すべき点はハードウェアの小型実装であり、暗号化回路, 復号回路, 鍵スケジュール回路を含むハードウェアを 0.18μm CMOS ASIC ライブラリを用いて 8.12K ゲートで実現できる。これは、既存の 128 ビットブロック暗号の中で最小クラスである。

もくじ

1　はじめに

日本電信電話株式会社 (以下では NTT と略す) と三菱電機株式会社 (以下では三菱電機と略す) が共同で開発した 128 ビットブロック暗号 *Camellia* を提案する。 Camellia では、ブロック長が 128 ビット、鍵長が 128 または 192 または 256 ビットであり、次期米国政府標準暗号 (以下 AES; Advanced Encryption Standard) のインタフェースに準拠している。また、設計目標は以下のとおりである。

高い安全性　最近の暗号解読技術の進展には目を見張るものがある。このため、差分攻撃 [BS93] や線形攻撃 [M94] に代表される強力な暗号解読技術に対する安全性を定量的に評価することが新しい暗号を設計する上で必要不可欠なことであると考えられている。我々は、最新の暗号解読技術を利用して Camellia の安全性を評価し、 2^{-128} 以上の確率を有するような差分特性や線形特性が Camallia には存在しないことを確認した。さらに、それら以外の攻撃法、例えば高階差分攻撃 [K95, JK97]、補間攻撃 [JK97, A00]、関連鍵攻撃 [B94, KSW96]、丸め差分攻撃 (truncated differential attacks) [K95, MT99]、ブーメラン攻撃 [W99]、スライド攻撃 [BW99, BW00] などに対しても安全であるように設計されている。

さまざまなプラットフォーム上での高い効率性　暗号システムはさまざまなアプリケーションで必要となるので、暗号アルゴリズムは幅広いプラットフォーム上で効率的に実装できることが望まれる。しかし、ソフトウェアとハードウェアの両面に適した 128 ビットブロック暗号アルゴリズムは数少ない。 Camellia は、さまざまなプラットフォーム上での処理速度とともに、ハードウェアでのゲート数やスマートカードでの使用メモリ量などを考慮して、ハードウェアとソフトウェア両面できわめて効率的な実装が可能となるように設計されている。

Camellia は、幅広いプラットフォーム上で効率的な実装が可能である 8 ビット入出力置換表 (s-box) と論理演算から構成されている。このため、低機能 IC カードで利用されている 8 ビット CPU から、PC で広く使われている 32 ビット CPU、さらには 64 ビット CPU まで、ソフトウエアで効率的な実装が可能である。また、 Camellia では、ソフトウェア実装に主眼を置いて設計されたいくつかの 128 ビットブロック暗号で広く使われている 32 ビット加算演算と乗算演算は使用しなかった。なぜなら、これらの算術演算は、 Pentium II/III や Athlon のような特殊なプラットフォーム上では高速に実行されるが、それ以外のプラットフォームでは高速な処理とはいえず、また、ハードウェア実装においては、長いクリティカルパスを構成し、かつ回路規模が大きくなる原因ともなるためである。

Camellia の置換表はハードウェア規模が最小になるように設計してある。つまり、 4 つの置換表は $\mathrm{GF}(2^8)$ 上の逆数関数のアフィン変換によって構成され、さらにその逆数関数はいくつかの $\mathrm{GF}(2^4)$

上の演算によっても実現できる。これにより、置換表をより少ないゲート数で実装することが可能となっている。

鍵スケジュールは非常に簡単な構造を有し、暗号化処理の一部分を共用している。また、動的 (on-the-fly) な副鍵生成が可能であり、そのとき暗号化・復号を問わず同じ効率で副鍵が生成される。副鍵生成のためのメモリ使用量も極めて小さく、128 ビット鍵では約 32 バイトの RAM、また 192 ビット鍵と 256 ビット鍵では約 64 バイトの RAM 使用量で実装できる。

標準化活動　2000 年 3 月、NTT と三菱電機は、国際標準として採用されることを目指し、ISO/IEC JTC 1/SC 27 からの公募に対して Camellia を提案した。また、2000 年 9 月、NESSIE (New European Schemes for Signature, Integrity, and Encryption) プロジェクトに対しても安全性の高い暗号要素として Camellia を提案し、2001 年 9 月に 2nd Phase に進む候補の一つとして選抜された。

本稿の構成　本稿の構成は以下のとおりである。第 2 章で Camellia の設計方針を、第 3 章で実装結果のまとめを述べる。第 4 章でソフトウェア実装における高速化手法を、第 5 章でハードウェア実装評価方法をそれぞれ示す。第 6 章で既知の攻撃に対する Camellia の安全性を述べる。最後に、第 7 章で本稿のまとめを記す。

Camellia の仕様については、別稿「Camellia 仕様書 – 128 ビットブロック暗号」を参照のこと。また、本稿中での定義や表記は「Camellia 仕様書」の記述に従っていることにも注意されたい。

2 設計指針

2.1 F 関数

Camellia のラウンド関数 (以下 F 関数) の設計指針は、 E2 の F 関数の設計指針 [KMA$^+$98] を踏襲している。 E2 と Camellia との主要な差異は、ラウンド関数の構造を 2 段換字置換網 (SPN; Substitution-Permutation Network) から 1 段換字置換網に変更した点である。 1 段換字置換網構造を Feistel 暗号のラウンド関数に利用したとき、差分特性確率や線形特性確率の上界値による理論的評価はより複雑になるものの、差分攻撃や線形攻撃に対する実際の安全性が同じ程度としたときの暗号化処理速度が改善されると期待される。この安全性評価については第 6 章で詳細を記しているので、そちらを参照されたい。

2.2 P 関数

Camellia の線形変換関数 (以下 P 関数) の設計方法は、 E2 の P 関数の設計方法 [KMA$^+$98] を踏襲している。すなわち、実装効率性の観点から排他的論理和演算 (XOR) のみで構成され、また差分攻撃や線形攻撃に対する安全性の観点から分岐数 (branch number) が最良となるような線形変換関数を P 関数の候補としている。さらに、 8 ビット CPU での実装のほか、 32 ビット CPU [AU00] および高機能 IC カードでの高速なソフトウェア実装方法を考慮して、上記の候補の中から 1 つの P 関数を選択した。

2.3 置換表

高い安全性およびハードウェア小型化の観点から、 GF(2^8) 上の逆数関数をアフィン変換した関数を利用して置換表を構成した。

GF(2^8) 上の関数における最大差分確率の最小値は 2^{-6} であることが証明されており、また最大線形確率の最小値も 2^{-6} となると予想されている。この 2^{-6} となる最良の最大差分確率と最大線形確率を達成する GF(2^8) 上の逆数関数をアフィン変換した関数が存在することから、これらの関数を置換表として採用した。このような置換表でのすべての出力ビットに対するブール多項式での次数は高いので、高階差分攻撃によって Camellia を解読することは困難である。また、 GF(2^8) 上の逆数関数の入出力において実行される 2 つのアフィン関数によって置換表の GF(2^8) 上での表現が複雑になり、補間攻撃も効率的ではなくなる。さらに、 4 つの異なる置換表を作ることによって、丸め差分攻撃 [MT99] に対する安全性が多少改善される。

ハードウェアの小型設計において、 GF(2^8) 上の要素は部分体 GF(2^4) 上の係数の多項式としても表現することができることから、 GF(2^4) 上の演算を何回か行うことによって置換表を実装すること

が可能である [MIYY88]。また、 $GF(2^8)$ 上の逆数関数の入出力において実行される 2 つのアフィン関数によって置換表の $GF(2^4)$ 上での表現も複雑になる。

2.4 FL 関数と FL^{-1} 関数

FL 関数と FL^{-1} 関数は、構造の規則性を崩すために Feistel 構造の 6 段ごとに挿入される。このような関数を挿入する目的の 1 つは、現在知られていない攻撃に対する防護を図ることである。 Feistel 構造の規則性が有する特徴のひとつに、暗号化処理と復号処理が副鍵の挿入順序を除いて同じ処理となっている点が挙げられる。そこで、 Camellia は FL 関数と FL^{-1} 関数とを 6 段ごとに挿入するものの、上記の特性を損なわないようにしている。

FL 関数とその逆関数である FL^{-1} 関数の設計指針は、 MISTY の FL 関数の設計指針を踏襲している [M97]。 MISTY と Camellia との差異は、 1 ビット循環シフトを加えたことである。これは、Camellia に対するバイト単位での暗号解読を難しくすることを意図すると同時に、ハードウェア規模や処理速度に対して負の影響が出ないようにするためのものである。これらの関数の設計方針は、固定された鍵に対しては (固定された) 線形変換になる一方で鍵の値によってその変換方法そのものが変わるようにすることである。すなわち、鍵が固定されている限り、これらの関数は (固定された) 線形変換となるので、平均差分確率や平均線形確率を大きくすることはない。さらに、論理積、論理和、排他的論理和 (XOR)、および循環シフトという論理演算によって構成されているので、ソフトウェアでもハードウェアでも高速な処理となっている。

2.5 鍵スケジュール

鍵スケジュールの設計指針は以下のとおりである。

1. 簡単な構成であり、さらに暗号化・復号の処理の一部分を共用できること。

2. 128 ビット鍵、 192 ビット鍵、 256 ビット鍵すべてについて副鍵生成回路が同様の鍵スケジュール回路で実行できること。さらに、 128 ビット鍵での鍵スケジュールはその回路の一部分を利用して実現できること。

3. 副鍵生成時間は暗号化処理時間よりも短いこと。
   1 つの秘密鍵で大量のデータを暗号化する場合には鍵セットアップ時間はあまり重要ではないかもしれない。しかし、秘密鍵が頻繁に変更されるようなアプリケーションでは鍵軽快性 (key agility) が重要となる。鍵軽快性の基本的要素の一つが副鍵生成時間である。

4. 動的 (on-the-fly) 副鍵生成が可能であること。

5. 動的 (on-the-fly) 副鍵生成時に、暗号化用の副鍵生成と復号用の副鍵生成の両方が同じ効率で計算できること
   いくつかの暗号では、暗号化用の副鍵生成と復号用の副鍵生成とが異なるものがある。また、Rijndael [DR98] や Serpent [ABK98] などのように、暗号化用の副鍵は逐次的に生成できるが、復号用の副鍵は最終の暗号化用副鍵から逆順に生成しなければならないような暗号もある。

6. 等価鍵が存在しないこと。

7. 関連鍵攻撃やスライド攻撃ができないこと。

指針 1 と 2 は主にハードウェアの小型実装を目的とした項目であり、指針 3、 4 および 5 は実際のアプリケーションにおける優位性を持たせるための項目である。指針 6 および 7 は安全性に関する項目である。

副鍵生成のために必要となるメモリ量は非常に小さい。 128 ビット鍵での Camellia の実装では、秘密鍵 K_L 用の 16 バイト (=128 ビット) と中間鍵 K_A 用の 16 バイト (=128 ビット) との合計 32 バイトが必要なメモリ量である。同様に、 192 ビット鍵および 256 ビット鍵の実装では合計 64 バイト必要となる。

3　実装評価

3.1　ソフトウェア実装

現在、もっともよく利用されている 32 ビットおよび 64 ビットプロセッサ上でのソフトウェア実装結果を表 1 に示す。また、メモリ量が制限されているような IC カードや組込みシステムで利用されるマイクロプロセッサ上でのソフトウェア実装結果を表 2 に示す。一般的に言えば，前者では “処理速度” が優先され，後者では “RAM や ROM の使用量” が重視される。データの中には公知のものもあるが [AIK+00a, C01, ISKM01, AIK+00b, Y01a, Y01b]、そうでないものも含まれている。

これらの表から Camellia は低機能 IC カード、 32 ビット、および 64 ビットプロセッサいずれでも効率よく実装できることがわかる。表中では 10^6 を M (mega)、 10^{-3} を m (milli) と略記した。

最適化の度合　アセンブリ言語を利用してプログラムを作成する時には、最良の結果が得られるように、第 4章で示した方法の多くを使うよう努力した。しかしながら、さらなる改善の余地が残っている。

一方、使用する C コンパイラによって、同じ C プログラムからでも異なるアセンブリコードが生成される。このことは、たとえ C プログラムが最適化されていたとしても、生成されたアセンブラコードが最適である保証がないことを意味する。したがって、 C プログラムの最適化に多くの時間は割かなかった。

どう速度計測するか　最近のプロセッサで速度計測するのは難しい。何故ならば、キャッシュの状態など制御が難しく速度に影響する要素がたくさんあるためである。我々は、速度計測にあたり、以下の状態などを仮定した。

- 全てのプログラムやデータは適切にワード境界等に配置されている。
- 暗号文と平文とプログラムは 1 次キャッシュに読み込まれている。
- 分岐予測が当たっている。
- 動的副鍵生成実装を除く、準備部関数は副鍵依存の定数を秘密鍵から生成し、その定数は暗号化又は復号関数で利用される。
- 動的副鍵生成実装を除く、暗号化または復号関数はブロック数の整数倍を暗号化又は復号する。
- キャッシュ外しやその他の制御不能要素を排除するため、何回も時間計測し最適なものを選んだ。

- 大きなブロックの暗号化時間を 1 ブロックあたりに平均した。但し、この数値は繰り返し処理や関数呼び出しの時間も含む。

3.2 ハードウェア実装

ASIC (Application Specific Integrated Circuit) 及び FPGA (Field Programmable Gate Array) 上でのハードウェア実装評価結果を表 3 に示す。ここでのハードウェア設計及び評価環境は表 4に示す通りである。

表 4: ハードウェア設計及び評価環境 (ASIC, FPGA)

記述言語	(ASIC, FPGA) Verilog-HDL
ライブラリ	(ASIC) 三菱電機 0.35μm CMOS ASIC ライブラリ 三菱電機 0.18μm CMOS ASIC ライブラリ 0.25 μm CMOS ASIC ライブラリ（暗号技術評価報告書 2000 年度版より） (FPGA) Xilinx XC4000XL シリーズ Xilinx VirtexE シリーズ
シミュレータ	(ASIC, FPGA) Verilog-XL （ASIC 0.25μm 用を除く） (ASIC) VCS5.1 (0.25μm 用)
論理合成ツール	(ASIC) Design Compiler バージョン 1998.08 (0.35μm 用) Design Compiler バージョン 2000.11-SP1 (0.18μm 用) Design Compiler バージョン 2000.05-1 (0.25μm 用) (FPGA) Synplify バージョン 5.3.1, ALLIANCE バージョン 2.1i (XC4000XL シリーズ用) Synplify バージョン 6.1.3, ALLIANCE バージョン 3.3.07i (VirtexE シリーズ用)

ここでは四つのタイプの回路 (Type 1 〜 Type 4) について評価を行った。それらの各タイプの回路の主な設計目標を表 5に示す。各回路タイプの詳細については第 5章を参照のこと。

表 5: ハードウェア設計方針 (概要)

タイプ	主な設計方針	回路概要
Type 1	暗号化 (復号) 処理において高速実装を実現する	図 1
Type 2	総回路規模において小型実装を実現する	図 2
Type 3	同上 (FPGA への適用に特化した場合)	図 3
Type 4	パイプライン実装を行う	図 4

表1: Camelliaのソフトウェア実装評価 (2001年8月31日更新)

プロセッサ	言語	鍵長	処理速度			RAM使用量(*1)		ROM使用量				参考文献
			鍵セットアップ(*2)	暗号化 / 復号		鍵セットアップ(*2)	暗号化/復号	合計	鍵セットアップ(*2)	暗号化/復号	表	
		[ビット数]	[サイクル数]	[サイクル数]	[Mbps]	[バイト数]	[バイト数]	[バイト数]	[バイト数]	[バイト数]	[バイト数]	
Pentium III (*4)	アセンブリ	128	1,570	308	290.9	288	20	15,012	6,788	0	8,224	[AIK+00b]
		128	160	371	241.5	28	36	11,420	1,046	2,150	8,224	[AIK+00b]
		192	222	494	181.4	28	36	13,032	1,469	3,323	8,240	[AIK+00b]
		256	226	494	181.4	28	36	13,048	1,485	3,323	8,240	[AIK+00b]
Pentium III (*5)	アセンブリ	128	-	326	255.2	-	-	29,285	-	-	-	[C01]
		128		(暗号化) 467 (*3)	0.72msec	-	-	20,110	-	-	-	[C01]
		128		(復号) 474 (*3)	0.73msec	-	-	20,236	-	-	-	[C01]
Pentium II (*6)	ANSI C(*7)	128	263	577	66.6	44	64	9,461	1,600	3,733	4,128	[AIK+00b]
Pentium III (*8)	Java(*9)	128	9,091	793	161.4	-	-	-	-	-	-	未公開
Alpha 21264 (*10)	アセンブリ	128	158	326	261.9	48	48	21,040	1,600	2,928	16,512	[AIK+00b]
		128	118	339	251.8	48	48	20,736	1,132	3,076	16,528	[AIK+00b]
		192	176	445	191.9	48	48	22,196	1,668	4,000	16,528	[AIK+00b]
		256	176	445	191.9	48	48	22,204	1,676	4,000	16,528	[AIK+00b]
Alpha 21264 (*11)	アセンブリ	128	-	282	210.2	-	-	31,552	-	-	-	[C01]
		128		(暗号化) 448 (*3)	0.97msec	-	-	25,792	-	-	-	[C01]
		128		(復号) 435 (*3)	0.94msec	-	-	25,792	-	-	-	[C01]
UltraSPARCIIi (*12)	アセンブリ	128	-	355	144.2	-	-	15,240	-	-	-	[C01]
		128		403 (*3)	1.01msec	-	-	23,992	-	-	-	[C01]

(*1) この欄の数字は, スタック領域を含むが, テキスト領域および鍵領域を除く。

(*2) 鍵スケジュールが含まれる場合もある。

(*3) この欄の数字は, 副鍵生成を含めた1ブロック暗号化の処理速度を示す。これは, 動的副鍵生成を利用して実現される。

(*4) IBM PC/AT互換PC, インテルPentium III (700MHz), on-die L2キャッシュ 256KB, FreeBSD 4.0R, 主記憶 128MB。

(*5) IBM PC/AT互換PC, インテルPentium III (650MHz), on-die L2キャッシュ 256KB, Windows98 SE, 主記憶 64MB。

(*6) IBM PC/AT互換PC, インテルPentium II (300MHz), L2キャッシュ 512KB, Windows95, 主記憶 160MB。

(*7) マイクロソフト Visual C++ 第6版。使用最適化オプション: **/G6 /Zp16 /ML /Ox /Ob2**。

(*8) IBM PC/AT互換PC, インテルPentium III (1GHz), on-die L2キャッシュ 256KB, Windows2000, 主記憶 512MB。

(*9) IBM Java Compiler 1.2.2ならびにIBM Java VM 1.2.2.。

(*10) Alpha 21264 (667MHz), Tru64 UNIX 4.0F, 主記憶 2GB。

(*11) Alpha 21264 (463MHz), Tru64 UNIX V5.1, 主記憶 512MB。

(*12) Ultra SPARC IIi (400MHz), Solaris 7, 主記憶 256MB。

表2: ICカードならびに組込システムでのプロセッサ上でのCamelliaのソフトウェア実装評価 (2001年8月31日更新)

プロセッサ	言語	鍵長	処理速度		RAM使用量		ROM使用量					参考文献
			鍵セットアップ(*1)	暗号化/復号	鍵セットアップ(*1)	暗号化/復号	合計	鍵セットアップ(*1)	暗号化/復号	表	共用サイズ(*2)	
		[ビット数]	[サイクル数]	[サイクル数]	[バイト数]	[バイト数]	[バイト数]	[バイト数]	[バイト数]	[バイト数]	[バイト数]	
8051 (*7)	アセンブリ	128	10,217 (*3) 10.22msec		0	32 (*4)	990	0	702	288	0	[AIK+00b]
Z80 (*8)	アセンブリ	128	5,146 1.03msec	28,382 5.68msec	44 (*5)	62 (*5)	1,698	358	1,183	288	-131	未公開
		128	(暗号化) 35,951 (*3) 7.19msec		0	60 (*5)	1,268	-	1,023	-	-797	未公開
			(復号) 37,553 (*3) 7.51msec		0	60 (*5)		-	1,042	-		
H8/3113 (*9)	アセンブリ	128	2,380 0.95msec	4,100 1.64msec	208 (*6)	0	-	-	-	-	-	[Y01a]
MC68HC705B16 (*10)	アセンブリ	128	7,500 3.57msec	9,900 4.71msec	208 (*6)	0	-	-	-	-	-	[Y01a]
MC68HC908AB32 (*11)	アセンブリ	128	5,679 0.71msec	8,430 1.05msec	208 (*6)	0	-	-	-	-	-	[Y01b]
M32Rx/D (*12)	アセンブリ	128	642 6.42msec	1,236 12.36msec	44 (*5)	44 (*5)	8,684	1,392	3,164	4,128	0	[AIK+00b]

(*1) 鍵スケジュールが含まれる場合もある。

(*2) 鍵スケジュール, 暗号化, 復号での各処理のうちいくつかの関数が共用できるので, ROM使用量が減少することがある。

(*3) この欄の数字は, 副鍵生成を含めた1ブロック暗号化の処理速度を示す。これは, 動的副鍵生成を利用して実現される。

(*4) この欄の数字は, スタック領域を含むが, テキスト領域および鍵領域を除く。

(*5) この欄の数字は, スタック領域, テキスト領域および鍵領域を含む。

(*6) この欄の数字は, 副鍵のサイズを示す。

(*7) Unix上のインテル8051 (12MHz; 1サイクル=12振動周期(oscillator periods))シミュレータ。

(*8) Windows上のZ80 (5MHz)シミュレータ。

(*9) 日立E6000エミュレータ上の日立H8/3113 (5MHz; 1サイクル=2振動周期(oscillator periods))。

(*10) モトローラIn-Circuitシミュレータキット上のモトローラ6805シリーズMC68HC705B16 (2.1MHz)。

(*11) モトローラIn-Circuitシミュレーションキット上のモトローラ6805シリーズMC68HC908AB32 (8MHz)。

(*12) MSA2310評価ボード上の三菱32ビットマイクロプロセッサM32Rx/D (100MHz)。

Table 3: Camelliaのハードウェア実装結果 (2001年8月31日更新)

アーキテクチャー			デザインコンパイラ	鍵長	処理速度				回路規模				効率性	参考文献
					鍵セットアップ	最大遅延(*1)	レイテンシ	スループット	単位	合計(*2)	副鍵生成(*3)	暗号化/復号(*4)	スループット/面積	
				[ビット数]	[nsec]	[nsec]	[サイクル数]	[Mbps]		[単位]	[単位]	[単位]	[Kbps/単位]	
ASIC	Unrolled	Type 1	三菱 0.35μm	128	24.36	109.35	1	1,170.55	Kgate	272.82	55.91	216.91	4.29	[AIK+00b]
			三菱 0.18μm	128	40.00	40.00	1	3,200.00		355.10	-	-	9.01	未公開
			三菱 0.18μm	128	45.96	45.96	1	2,785.00		244.90	-	-	11.37	未公開
	Loop	Type 2	三菱 0.35μm	128	110.20	27.67	21	220.28	Kgate	11.35	4.98	6.37	19.41	[AIK+00a]
			三菱 0.35μm	128	117.04	28.73	21	212.16		9.66	5.75	3.91	21.96	[AIK+00b]
			三菱 0.18μm	128	144.88	36.22	21	168.28		8.51	-	-	19.77	未公開
			三菱 0.18μm	128	25.92	6.48	21	940.62		27.46	-	-	34.25	未公開
			三菱 0.18μm	128	28.20	7.05	21	864.57		21.45	-	-	40.31	未公開
			三菱 0.18μm	128	23.20	5.80	21	1,050.90		11.87	-	-	88.52	未公開
			三菱 0.18μm	128	137.24	34.31	21	177.65		8.12	-	-	21.87	未公開
			三菱 0.18μm	128	12.96	3.24	21	1,881.25		44.30	-	-	42.47	未公開
			(匿名) 0.25μm	256	-	5.46	-	837.00		39.35	22.76	16.33	21.27	[C01]
			(匿名) 0.25μm	256	-	11.51	-	397.00		23.12	13.30	9.67	17.17	[C01]
FPGA	Loop	Type 2	Xilinx XC4000XL	128	362.83	78.82	21	77.34	CLB	1,296	-	-	59.68	未公開
		Type 3	Xilinx XC4000XL	128	-	50.00	21	122.01		874	-	-	139.60	未公開
		Type 2	Xilinx VirtexE	128	135.03	30.56	21	199.46	Slice	1,816	-	-	109.83	[ISKM01]
			Xilinx VirtexE	128	126.00	28.80	21	211.90		1,816	-	-	116.69	[ISKM01]
			Xilinx VirtexE	128	127.04	26.80	21	227.42		1,780	-	-	127.76	[ISKM01]
	Unrolled	Type 1	Xilinx VirtexE	128	97.70	318.50	1	401.89		9,426	-	-	42.64	[ISKM01]
	Pipeline	Type 4	Xilinx VirtexE	128	83.25	18.96	20	6,749.99		9,692	-	-	696.45	[ISKM01]

(*1) 暗号化(または復号)回路でのクリティカルパス。

(*2) この欄の数字は, 副鍵生成回路, 暗号化／復号回路, (必要ならば)データセレクタ, 出力レジスタ, 副鍵レジスタ及びファンアウト調整用バッファを含む。

(*3) この欄の数字は, 副鍵レジスタを含む。

(*4) この欄の数字は, 出力レジスタと(必要ならば)データセレクタを含む。

4　効率的なソフトウェア実装法

この章では、 Camellia をソフトウェアで効率的に実装する方法を説明する。ほとんどの場合、プログラムは**準備部** (鍵スケジュールを含む) と**データ撹拌部** (暗号化又は復号) の 2 つの部分から構成される。まず準備部の最適化法を説明し、次にデータ撹拌部の最適化法を説明する。

この章では、 8、 32、 64 ビットプロセッサ向けの特別な方法も説明する。しかしながら、 8 ビットプロセッサ向けの最適化法は 32 や 64 ビットプロセッサにも適用可能であり、 32 ビットプロセッサ向けの最適化法は 64 ビットプロセッサにも適用可能である。他の語長も必要に応じて考慮されたい。

この章では、まず Camellia が仕様通りに実装されていると仮定している。その後、その実装をどう最適化するかについて論じる。

この章では語長は対象プロセッサの自然な大きさを指す。例えば MMX なしの IA-32 では 32 ビット、 MMX ありの IA-32 や Alpha では 64 ビットを指す。

4.1　準備部

4.1.1　全ての副鍵の記憶

もし、十分にメモリがあれば、一旦生成した副鍵をメモリに保存し、データ撹拌部でそれを利用する。

4.1.2　副鍵の生成順序

副鍵は必ずしも利用される順に生成しなくてもよい。例えば、 128 ビット鍵の場合、まず K_L に依存する副鍵だけを生成し、次に K_A のみに依存する副鍵を生成する。そうすれば、 K_A を保持しておくためのレジスタか、またはメモリを節約することができる。

4.1.3　鍵スケジュール中の排他的論理和による相殺

Camellia の鍵スケジュールは Feistel 構造に基づいている。 2 段目と 3 段目の間で K_L が中間値と排他的論理和される。この構造は K_L の相殺を引き起こす。正確には、 3 段目の入力は以下の式により計算できる。

$$\begin{cases} (\text{右半分}) &= F(K_{LL}, \Sigma_1) \\ (\text{左半分}) &= F(K_{LR} \oplus (\text{右半分}), \Sigma_2) \end{cases} \quad \text{128 ビット鍵}$$

$$\begin{cases} (\text{右半分}) &= K_{RR} \oplus F(K_{LL} \oplus K_{RL}, \Sigma_1) \\ (\text{左半分}) &= K_{RL} \oplus F(K_{LR} \oplus (\text{右半分}), \Sigma_2) \end{cases} \quad \text{192 ビット鍵と 256 ビット鍵}$$

上式を用いれば、128 ビット鍵の場合は 3 回、192 または 256 ビット鍵の場合は 2 回の **L** 上の排他的論理和を仕様通りの実装法に比べて削減することができる。

4.1.4 K_L, K_R, K_A, K_B の循環ビット数

副鍵生成時に、K_L, K_R, K_A, K_B について、循環シフトした値を保持しておけば元の値を保持しておく必要はない。副鍵はこの保持した値を 16 ± 1 の和だけ循環シフトすれば生成できる。

4.1.5 k_{11} と k_{12} から kl_5 と kl_6 の生成

192 と 256 ビット鍵では、(kl_5, kl_6) は $(k_{11}, k_{12}) \lll_{32}$ に等しいので (k_{11}, k_{12}) から (kl_5, kl_6) の生成にワード処理の循環シフトを利用できる。この事実を用いると、一般的な循環シフト方法に比べて、数命令削減できる。

4.1.6 動的副鍵生成

副鍵は動的に生成することが可能である。全ての副鍵は K_L, K_R, K_A, K_B のいずれかを循環シフトしたものである。従って、最初に K_L, K_R, K_A, K_B を生成し、次にそれらを循環シフトすればよい。K_L, K_R, K_A, K_B の循環シフトビット数については、第 4.1.4章を参照のこと。

4.1.7 128 ビット鍵と 192・256 ビット鍵

もし、128 ビット鍵より長い鍵長が不要であるならば K_B を生成する必要はない。つまり、最後の 2 つの F 関数の計算を省略することができる。

4.1.8 Q 上の元をどう循環シフトするか

8 ビットプロセッサ 第 4.1.4章に述べたように、循環シフト量の合計は 16 ± 1 の倍数の和である。16 ± 1 ビット循環シフトは 2 バイト移動の後に 1 ビットシフトすることにより実現できるので、**Q** 上の元の循環シフトは効率的に実装できる。

32 ビットプロセッサ もし IA-32 上の実装ならば、倍精度シフト命令である `shrd` や `shld` の利用を考慮されたい。

4.1.9 F 関数

鍵スケジュールは F 関数を含んでいるが、F 関数は主にデータ攪拌部で用いられる。第 4.2章を参照のこと。

4.1.10 鍵依存関数

Camellia は排他的論理和と論理和と論理積の 3 つの鍵依存命令を使っている。もし可能であるならば、これらの命令に対する自己書き換えプログラムの利用も検討されたい。

4.2 データ攪拌部

4.2.1 Endian 変換

Camellia は big endian を基本に構成されている。従って、 little endian などの big endian でないプロセッサでの実装は endian 変換のためのちょっとしたプログラムの追加が必要である。

もっとも安易な実装法としてはメモリからレジスタに読み込むときと、レジスタからメモリに書き出す際に endian 変換を行なう方法がある。しかしながら、 FL 関数と FL^{-1} 関数のみが endian に依存している。より正確には、 FL 関数や FL^{-1} 関数中の 1 ビット循環シフトのみが endian 依存である。つまり、副鍵生成法を適切に調整すれば、 endian 変換を 1 ビット循環シフトとその直後に行なえばよい。 1 ビット循環シフトと endian 変換を組み合わせた計算を行なうと Camellia の性能を向上できるかもしれない。この方針については、第 4.2.2章に詳細を記した。

いくつかのプロセッサは endian 変換のための特別な命令を持っている。例えば、 80486 以降の IA-32 は `bswap` を具備している。これらの命令を使用されたい。但し [C98, Appendix A] に書いてある endian 変換方法は使ってはいけない。この方法によりプログラム規模は小さくなるが、メモリ読み込みや書き込みは遅延が大きいので高速にならない。

上に記した様に、 endian の問題は 32 ビットワードの 1 ビット循環シフトのみに影響する。従って、 64 ビットワード全ての endian 変換をする必要はない。

以下に 32 ビットレジスタ x に対する一般的な endian 変換法を示す。下記の方法で、式中の $+$ の代わりに $\cup$ や $\oplus$ を、また、適切に mask 定数を変換すればシフトや循環シフトと論理和の計算順序を変換できることに注意されたい。

真正直な方法

$$x \leftarrow (x \lll_{24}) + ((x \cap \texttt{0xff00}) \lll_{8}) + ((x \ggg_{8}) \cap \texttt{0xff00}) + (x \ggg_{24})$$

この方法は並列度が高い。

循環シフトを使わない場合の命令数最小の方

$$\begin{aligned} x &\leftarrow (x \lll_{16}) + (x \ggg_{16}) \\ x &\leftarrow ((x \cap \texttt{0xff00ff}) \lll_{8}) + ((x \ggg_{8}) \cap \texttt{0xff00ff}) \end{aligned}$$

循環シフトを利用する方

$$x \leftarrow ((x \cap \texttt{0xff00ff}) \ggg_8) + ((x \lll_8) \cap \texttt{0xff00ff})$$

SSE の利用　Pentium III を含むインテルの新しいプロセッサ群は `pshufw` というデータの並び変えに非常に有効な命令が使える [I99]。`pshufw` を使うと 64 ビットデータの endian 変換はわずか 5 命令 (μops) で実現できる。

4.2.2　Little endian 解釈の 1 ビット循環シフト

第 4.2.1章に書いたように、もし FL 関数や FL^{-1} 関数中の 1 ビット循環シフトを効率良く実装できるのであれば、平文や暗号文を読み込んだり書き出したりするときの endian 変換は必要ない。

ここで x を 1 ビット循環シフトすべき little endian でデータが格納されている 32 ビット長のレジスタとする。このとき x の 1 ビット循環シフトは以下の式で計算できる。

$$x \leftarrow ((2x) \cap \texttt{0xfefefefe}) + ((x \ggg_{15}) \cap \overline{\texttt{0xfefefefe}}) \tag{1}$$

もちろん、この方法を使うには副鍵生成や他の関数群も適切に修正する必要がある。

式 (1) 中の $+$ は $\cup$ または $\oplus$ に置き換えてもよく、また、 $2x$ の計算は $\ll_1$ または $\lll_1$ または x 自身への加算に置き換えてもよい。さらに、適切に mask を変えればシフトや循環シフトと論理和の順序を入れ換えることができる。

IA-32 の `pandn` や、 Alpha の `bic` の様な否定論理積 (ANDNOT) が Camellia を実装しようとしているプロセッサにあるかどうかを確認されたい。この場合は、定数 $\overline{\texttt{0xfefefefe}}$ を準備する必要はない。

4.2.3　初期及び最終排他的論理和 (whitening)

下記の式を用いれば kw_2 と kw_4 との加算は他の鍵加算に統合できる。

$$\begin{aligned}
(x \oplus k) \oplus y &= (x \oplus y) \oplus k, \\
(x \oplus k) \oplus l &= x \oplus (k \oplus l), \\
(x \oplus k) \cap l &= (x \cap l) \oplus (k \cap l), \\
(x \oplus k) \lll_1 &= (x \lll_1) \oplus (k \lll_1), \\
(x \oplus k) \cup l &= (x \cup l) \oplus (k \cap \bar{l}),
\end{aligned} \tag{2}$$

ここで、 x, y, k, l はビット列とする。この副鍵変換を準備部で行なえば $\mathbf{L}$ 上の 2 回の排他的論理和を省略できる。

4.2.4 副鍵との排他的論理和

式 (2) を用いれば、 S 関数を通過しない範囲で副鍵との排他的論理和の順序を任意の位置に変更できる。例えば、 F 関数の計算を $P(S(X \oplus k))$ から $P(S(X)) \oplus k'$ と変更すれば、命令の依存性が改善されるかもしれない。

4.2.5 S 関数

s_1 は GF(2^8) 上の四則演算により定義されている。しかしながら、 GF(2^8) 上の演算を計算してはいけない。代わりに事前計算し、その結果をプログラム中に埋め込むべきである。仕様の表 4 を参照のこと。

もし十分なメモリがあり、かつ 8 ビットデータの循環シフトが高コストならば、 s_2, s_3, s_4 の表も s_1 に加えて事前計算し、プログラムに埋め込むことを強く推奨する。もし、十分なメモリがないのであれば、 s_2, s_3, s_4 は s_1 の表と 1 回の循環シフトを使って計算されたい。 (仕様の第 4.5 章参照のこと)

Java の仮想計算機 (virtual machine) のように表引き処理が重くまた十分なメモリがある環境であれば 2 つの置換表を合成した表、例えば $(s_1(y_1), s_2(y_2))$ の利用も考慮されたい。

4.2.6 P 関数

32 ビットプロセッサ $(Z_L, Z_R) = ((z_1, z_2, z_3, z_4), (z_5, z_6, z_7, z_8))$ を P 関数の入力、 $(Z'_L, Z'_R) = ((z'_1, z'_2, z'_3, z'_4), (z'_5, z'_6, z'_7, z'_8))$ を P 関数の出力とする。

仕様の図 5 から P 関数は以下のように計算できることがわかる。

$$\begin{aligned}
Z_L &\leftarrow Z_L \oplus (Z_R \lll_8) \\
Z_R &\leftarrow Z_R \oplus (Z_L \lll_{16}) \\
Z_L &\leftarrow Z_L \oplus (Z_R \ggg_8) \\
Z_R &\leftarrow Z_R \oplus (Z_L \ggg_8) \\
Z'_L &\leftarrow Z_R \\
Z'_R &\leftarrow Z_L
\end{aligned}$$

この計算の命令依存性は高い。この計算は以下のように変更できる。

$$
\begin{array}{lllllll}
 & & & Z_R & \leftarrow & Z_R \lll_8 \\
Z_L & \leftarrow & Z_L \oplus Z_R & Z_R & \leftarrow & Z_R \lll_8 \\
Z_L & \leftarrow & Z_L \ggg_8 & Z_R & \leftarrow & Z_R \oplus Z_L \\
Z_L & \leftarrow & Z_L \oplus Z_R & Z_R & \leftarrow & Z_R \lll_{16} \\
Z_L & \leftarrow & Z_L \lll_8 & Z_R & \leftarrow & Z_R \oplus Z_L \\
Z'_L & \leftarrow & Z_R & Z'_R & \leftarrow & Z_L
\end{array}
$$

上記計算法の命令依存度は減少している。この方法は 1 回余分な循環シフトを必要とするように見えるが、大抵の場合、最初の計算と S 関数の計算を追加命令なしで同時に計算することができるので、実質的な命令数増加はないであろう。

8 ビットプロセッサ (直交命令系) もし、排他的論理和が任意のレジスタの組合せで計算でき、十分な数のレジスタがあるならば、仕様の図 5 に書いてある方法で、 P 関数を 16 回の排他的論理和で計算できる。

8 ビットプロセッサ (累算器型) もし累算器型のプロセッサでの実装を考えている場合、排他的論理和数を最小化することは必ずしも良い考えとは言えない。何故ならば排他的論理和数を最小化するに当たってメモリからレジスタへの読み込みや、レジスタからメモリへの書き込みが多数必要となることがあるからである。以下の計算は、累算器型プロセッサのために最適化してある。

$$
\begin{array}{lll}
z'_8 & \leftarrow & z_1 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7 \\
z'_4 & \leftarrow & z'_8 \oplus z_1 \oplus z_2 \oplus z_3 \\
z'_7 & \leftarrow & z'_4 \oplus z_2 \oplus z_7 \oplus z_8 \\
z'_3 & \leftarrow & z'_7 \oplus z_1 \oplus z_2 \oplus z_4 \\
z'_6 & \leftarrow & z'_3 \oplus z_1 \oplus z_6 \oplus z_7 \\
z'_2 & \leftarrow & z'_6 \oplus z_1 \oplus z_3 \oplus z_4 \\
z'_5 & \leftarrow & z'_2 \oplus z_4 \oplus z_5 \oplus z_6 \\
z'_1 & \leftarrow & z'_5 \oplus z_2 \oplus z_3 \oplus z_4
\end{array}
$$

もし、 z'_i の添字調整が重い処理の場合は、以下の方法が有効である。

$$
\begin{array}{lll}
\sigma & \leftarrow & z_1 \oplus z_2 \oplus z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7 \oplus z_8 \\
z'_1 & \leftarrow & \sigma \oplus z_2 \oplus z_5
\end{array}
$$

$$
\begin{aligned}
z'_2 &\leftarrow \sigma \oplus z_3 \oplus z_6 \\
z'_3 &\leftarrow \sigma \oplus z_4 \oplus z_7 \\
z'_4 &\leftarrow \sigma \oplus z_1 \oplus z_8 \\
z'_5 &\leftarrow \sigma \oplus z_3 \oplus z_4 \oplus z_5 \\
z'_6 &\leftarrow \sigma \oplus z_1 \oplus z_4 \oplus z_6 \\
z'_7 &\leftarrow \sigma \oplus z_1 \oplus z_2 \oplus z_7 \\
z'_8 &\leftarrow \sigma \oplus z_2 \oplus z_3 \oplus z_8
\end{aligned}
$$

4.2.7 換字置換

この章では S と P を独立に計算するより効率良く $P \circ S$ を計算する方法について論じる。

64 ビットプロセッサ　もし、実装を考えているプロセッサの 1 次キャッシュが十分に大きいのであれば、文献 [RDP+96] の方法を使用されたい。この方法では、式 (3) で定義される表を用いる。

$$
\begin{array}{lcl}
SP_1(y_1) &=& (s_1(y_1),\ s_1(y_1),\ s_1(y_1),\ 0,\ s_1(y_1),\ 0,\ 0,\ s_1(y_1)) \\
SP_2(y_2) &=& (0,\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ s_2(y_2),\ 0,\ 0) \\
SP_3(y_3) &=& (s_3(y_3),\ 0,\ s_3(y_3),\ s_3(y_3),\ 0,\ s_3(y_3),\ s_3(y_3),\ 0) \\
SP_4(y_4) &=& (s_4(y_4),\ s_4(y_4),\ 0,\ s_4(y_4),\ 0,\ 0,\ s_4(y_4),\ s_4(y_4)) \\
SP_5(y_5) &=& (0,\ s_2(y_5),\ s_2(y_5),\ s_2(y_5),\ 0,\ s_2(y_5),\ s_2(y_5),\ s_2(y_5)) \\
SP_6(y_6) &=& (s_3(y_6),\ 0,\ s_3(y_6),\ s_3(y_6),\ s_3(y_6),\ 0,\ s_3(y_6),\ s_3(y_6)) \\
SP_7(y_7) &=& (s_4(y_7),\ s_4(y_7),\ 0,\ s_4(y_7),\ s_4(y_7),\ s_4(y_7),\ 0,\ s_4(y_7)) \\
SP_8(y_8) &=& (s_1(y_8),\ s_1(y_8),\ s_1(y_8),\ 0,\ s_1(y_8),\ s_1(y_8),\ s_1(y_8),\ 0)
\end{array}
\tag{3}
$$

次に、以下の式を計算する。

$$
(z'_1, z'_2, z'_3, z'_4, z'_5, z'_6, z'_7, z'_8) \leftarrow \bigoplus_{i=1}^{8} SP_i(y_i)
$$

この方法の計算量は次の通り。

表参照回数	8
排他的論理和数	7
表の大きさ (KB)	16

もし、プロセッサの 1 次キャッシュがそれなりに大きい場合は、式 (3) で定義される表のいくつかを下記の表に置き換える。

$$\begin{aligned}
SP_\alpha(y) &= (s_1(y),\, s_1(y),\, s_1(y),\, s_1(y),\, s_1(y),\, s_1(y),\, s_1(y),\, s_1(y)) \\
SP_\beta(y) &= (s_2(y),\, s_2(y),\, s_2(y),\, s_2(y),\, s_2(y),\, s_2(y),\, s_2(y),\, s_2(y)) \\
SP_\gamma(y) &= (s_3(y),\, s_3(y),\, s_3(y),\, s_3(y),\, s_3(y),\, s_3(y),\, s_3(y),\, s_3(y)) \\
SP_\delta(y) &= (s_4(y),\, s_4(y),\, s_4(y),\, s_4(y),\, s_4(y),\, s_4(y),\, s_4(y),\, s_4(y))
\end{aligned} \tag{4}$$

次に、必要なバイト位置に 0 を埋め込む。もし式 (4) のみを利用した場合は、次の計算量を要する。

表参照回数	8
排他的論理和数	7
論理和数	8
表の大きさ (KB)	8

この方法を Alpha アーキテクチャ [C98] 上で実装しており、適切なバイト位置に 0 を埋め込むためのマスク定数を保持するためのレジスタ数が十分でないなら、 `zap` または `zapnot` を利用されたい。

もし、利用するプロセッサが IA-32[I99] にある Pentium with MMX technology 以降で採用された `punpckldq` 命令や、 Pentium II 以降で採用された `pshufw` 命令の様に、レジスタの半分のビットを残りの半分に効率的に複写できるのであれば、式 (3) 中の SP_1, SP_2, SP_3, SP_4 のみを準備する。そして、以下の式を計算する。

$$\begin{aligned}
&(z'_1, z'_2, z'_3, z'_4, z'_5, z'_6, z'_7, z'_8) \\
&\leftarrow SP_1(y_1) \oplus SP_2(y_2) \oplus SP_3(y_3) \oplus SP_4(y_4) \oplus \nu(SP_1(y_8) \oplus SP_2(y_5) \oplus SP_3(y_6) \oplus SP_4(y_7)),
\end{aligned}$$

ここで、 ν は最初の 4 バイトを後の 4 バイトに複写する演算とする。この方法は次の計算量を要する。

表参照回数	8
排他的論理和数	7
ν の回数	1
表の大きさ (KB)	8

32 ビットプロセッサ　文献 [AU00] には Camellia 型の換字置換網の効率的な実装法が示されている。そのうちの一つの方法は、まず次に示す式 (5) を準備する。

$$\begin{aligned}
SP_{1110}(y) &= (s_1(y),\, s_1(y),\, s_1(y),\, 0) \\
SP_{0222}(y) &= (0,\, s_2(y),\, s_2(y),\, s_2(y)) \\
SP_{3033}(y) &= (s_3(y),\, 0,\, s_3(y),\, s_3(y)) \\
SP_{4404}(y) &= (s_4(y),\, s_4(y),\, 0,\, s_4(y))
\end{aligned} \tag{5}$$

そして、次の式を計算する。

$$\begin{aligned} D &\leftarrow SP_{1110}(y_8) \oplus SP_{0222}(y_5) \oplus SP_{3033}(y_6) \oplus SP_{4404}(y_7) \\ U &\leftarrow SP_{1110}(y_1) \oplus SP_{0222}(y_2) \oplus SP_{3033}(y_3) \oplus SP_{4404}(y_4) \\ (z'_1, z'_2, z'_3, z'_4) &\leftarrow D \oplus U \\ (z'_5, z'_6, z'_7, z'_8) &\leftarrow (z'_1, z'_2, z'_3, z'_4) \oplus (U \ggg_8) \end{aligned}$$

この方法は、次に示す計算量を要する。

表参照回数	8
排他的論理和数	8
循環シフト	1
表の大きさ (KB)	4

文献 [AU00] は、循環シフト処理が非常に重いプロセッサのための実装法も示している。この方法では、式 (5) で定義される表に加え次の式で定義される表も準備する。

$$\begin{aligned} SP_{1001}(y) &= (s_1(y), \quad 0, \quad 0, s_1(y)) \\ SP_{2200}(y) &= (s_2(y), s_2(y), \quad 0, \quad 0) \\ SP_{0330}(y) &= (\quad 0, s_3(y), s_3(y), \quad 0) \\ SP_{0044}(y) &= (\quad 0, \quad 0, s_4(y), s_4(y)) \end{aligned}$$

そして、次の式を計算する。

$$\begin{aligned} D &\leftarrow SP_{1110}(y_8) \oplus SP_{0222}(y_5) \oplus SP_{3033}(y_6) \oplus SP_{4404}(y_7) \\ (z'_1, z'_2, z'_3, z'_4) &\leftarrow D \oplus SP_{1110}(y_1) \oplus SP_{0222}(y_2) \oplus SP_{3033}(y_3) \oplus SP_{4404}(y_4) \\ (z'_5, z'_6, z'_7, z'_8) &\leftarrow D \oplus SP_{1001}(y_1) \oplus SP_{2200}(y_2) \oplus SP_{0330}(y_3) \oplus SP_{0044}(y_4) \end{aligned}$$

この方法は次の計算量を要する。

表参照回数	12
排他的論理和数	11
表の大きさ (KB)	8

4.2.8 置換表の引数生成

置換表の引数は単純にシフトと論理積を用いることにより生成できる。しかしながら、いくつかのプロセッサはこの引数を生成するための特別な命令を持っている。例えば、IA-32[I99] の `movzx` や、Alpha[C98] の `extbl` がある。

P6 で `movzx` は高速な演算であるが、下位 2 バイトに対してしか用いられない。素朴な方法として `eax`, `ebx`, `ecx`, `edx` レジスタを (L_r, R_r) の保持に用い、 4 回の循環シフトを毎段実行する実装が考えられる。循環シフトのうち 2 回は、レジスタ中のバイト位置を元に戻すために必要である。しかしながら、もし循環シフトされた表を準備するのであれば毎段のバイト位置を補正するための 2 回の循環シフトを省略できる。この実装法によると、バイト位置は 4 段毎に復帰することに注意されたい。

4.3 一般的な注意事項

この章では、一般的な注意事項について述べる。これらの注意事項は、他のブロック暗号を最適化するのと同様 Camellia の最適化に際しても有効である。より詳しくは、それぞれのプロセッサに対する最適化のための取扱説明書を参照のこと。

データの整列 (align) 整列されていないデータの利用はほとんどのプロセッサにおいて、非常に重い処理である。データはワード境界に整列すべきである。

部分データの利用の回避 ほとんどのプロセッサでは、バイト処理などのワードより小さな大きさを利用するための機能を持っている。しかしながらこの機能はしばしば重い処理となっている。もし、十分なメモリを使えるのであれば、ワードの大きさが必要でないとしても、ワード利用のみにすべきである。

キャッシュの大きさへの注意 もし、プログラムやデータの大きさがキャッシュの大きさを越えると、実行速度が急激に遅くなる。ループ展開や表展開は高速化のためのよい方法であるが、キャッシュの大きさを越えないよう注意すべきである。

組み込み (intrinsic) 関数の使用 いくつかのコンパイラでは組み込み関数を利用できる。例えば、 IA-32 上のマイクロソフトの Visual C++ 第 6 版では、「`#pragma intrinsic(_lrotl)`」と宣言し、「`_lrotl`」を利用すればコンパイラはアセンブリ言語レベルで循環シフト命令を生成する。詳細については利用しているコンパイラの取扱説明書を参照のこと。

正確な速度計測は困難 プログラムの実行速度はキャッシュ外しや、 OS 割り込みなどの非常に多くの要因に依存する。さらに、何ブロック暗号化したかなどの暗号学的な要素も実行時間に影響する。

いくつかのプロセッサにはプロセッサ起動時からの経過サイクル数を得る命令がある。例えば Pentium 以降の IA-32 は `rdtsc` 命令があり [I99]、 Alpha には `rpcc` 命令がある [C98]。この様な命令を速度計測に用いることは良い考えであるが P6 や EV6 の様な非逐次実行 (out-of-order) アーキテクチャにはそのまま適用してはいけない。

もし、正確な実行時間を計測したいのであれば適切な案内書を参照されたい。例えば、 Pentium 系のプロセッサについては文献 [F00] を参照のこと。

5 ハードウェア実装評価

第3章において、 Camellia のハードウェア実装評価結果 (ASIC 及び FPGA) を示した。本章では、第3章で示した4タイプの回路の設計方針について述べる。以下に各々のタイプの詳細を示す。

5.1 Type 1: 高速実装-1 (フーリー・ループ・アンロールドアーキテクチャ)

Type 1では、回路規模に関しては制限を持たせずに、暗号化及び復号処理においてできるだけ高速実装を実現するような場合 (ASIC 及び FPGA) について検討する。図1は、 Type 1 回路の概要である。表6に Type 1 回路の基本コンポーネントを示す。

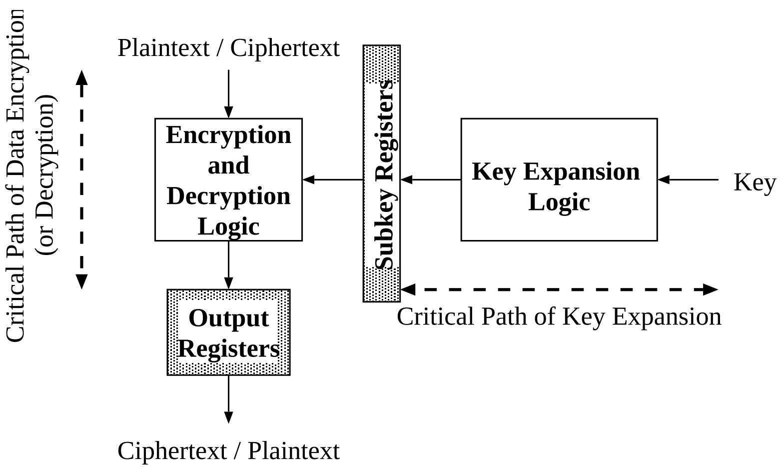


図 1: Type 1 回路の概要 (ASIC, FPGA)

表 6: Type 1 回路の基本コンポーネント

暗号化及び復号回路	組合せ回路より構成された暗号化及び復号のためのデータ撹拌部回路
出力レジスタ	暗号化 (復号) データのためのレジスタ
鍵拡大回路	組合せ回路より構成された暗号化鍵から副鍵を生成する回路
副鍵レジスタ	鍵拡大回路部の出力データのためのレジスタ

これらの基本コンポーネントの設計方針を以下に示す。

1. “暗号化及び復号回路” 及び “鍵拡大回路”

   (a) ループアーキテクチャは採用しない。

   (b) パイプラインアーキテクチャは採用しない。

   (c) 置換表は人手で最適化せず、論理合成ツールのみを用いて最適化を行う。

2. “出力レジスタ” 及び “副鍵レジスタ”

   (a) 出力レジスタのサイズは 1 ブロック分 (=128 ビット) とする。

   (b) 副鍵レジスタのサイズはアルゴリズム内の全副鍵の長さ分とする。

以上の設計方針のもとで、 Camellia を ASIC 及び FPGA に実装した場合について評価検討を行った。それらの評価結果は第 3章において表 3 に示したとおりである。ここで “スループット (Throughput)” は次の式で定義されるものとする:

$$\text{スループット [b/s]} = \frac{\text{ブロックサイズ (128 [bits])}}{\text{暗号化 (復号) 回路のクリティカルパス [sec]}}$$

5.2 Type 2: 小型実装 -2 (ループアーキテクチャ)

Type 2 では、暗号化及び復号処理においてできるだけ小型実装を実現するような場合 (ASIC 及び FPGA) について検討する。図 2は Type 2 回路の概要である。表 7に Type 2 回路の基本コンポーネントを示す。

表 7: Type 2 回路の基本コンポーネント

暗号化及び復号回路	組合せ回路から構成され、鍵拡大回路 (の一部) を含む一段分の暗号化及び復号のためのデータ撹拌部回路
出力レジスタ	出力 (および中間処理状態の) データのためのレジスタ
データセレクタ	暗号化 (復号) データまたは出力データを選択するセレクタ
鍵スケジュール回路	組合せ回路から構成され、暗号化及び復号回路部の中にある鍵拡大回路 (の一部) を用いて副鍵を生成する回路
副鍵レジスタ	鍵スケジュール回路部の出力データのためのレジスタ

これらの基本コンポーネントの設計方針を以下に示す。

1. “暗号化及び復号回路” 及び “鍵スケジュール回路”

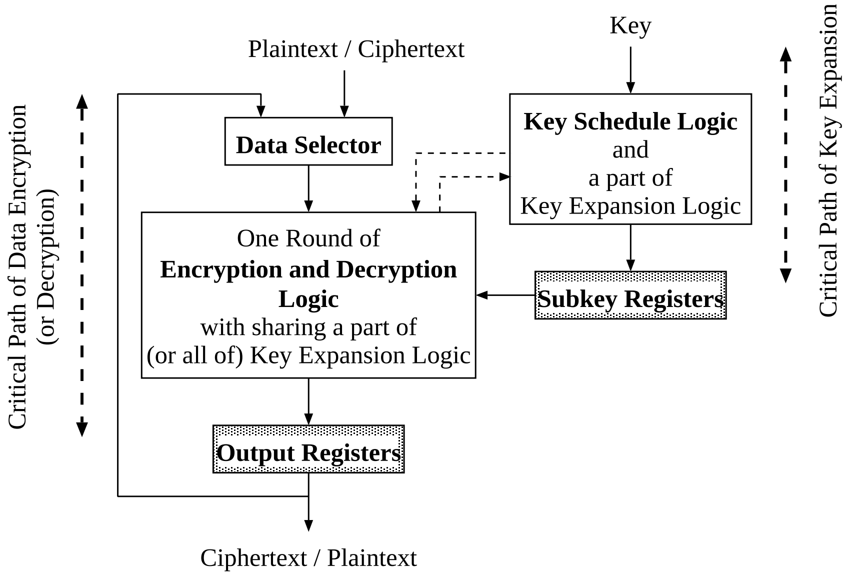


図 2: Type 2 回路の概要 (ASIC, FPGA)

(a) ループアーキテクチャを採用する。

(b) パイプラインアーキテクチャは採用しない。

(c) 置換表は設計者の手により最適化の後、論理合成ツールを用いて最適化を行う。

(d) 鍵スケジュール回路部は、鍵拡大回路 (の一部) と制御回路から構成される。

2. "出力レジスタ"、"副鍵レジスタ" 及び "データセレクタ"

(a) 出力レジスタのサイズは一ブロック分 (=128 ビット) とする。

(b) 副鍵レジスタのサイズは暗号化復号ロジック部で用いられる副鍵のサイズとする。

(c) データセレクタは 1 ブロック分 (=128 ビット) の 2-1 セレクタとする。

以上の設計方針のもとで、 Camellia を ASIC 及び FPGA に実装した場合について評価検討を行った。それらの評価結果は第 3章において表 3 に示したとおりである。ここで "スループット (Throughput)" は次の式で定義されるものとする:

$$\text{スループット [b/s]} = \frac{\text{ブロックサイズ (128 [bits])}}{\text{暗号化 (復号) 回路のクリティカルパス [sec]} \times \text{レイテンシ}}$$

5.3 Type 3: 小型実装 -2 (FPGA への適用に特化した場合, ループアーキテクチャ)

Type 3 では、暗号化及び復号処理においてできるだけ小型実装を実現するような Type 2 における特殊な場合 (全ての副鍵を外部から入力し副鍵全てをメモリに蓄える方式。ここでは FPGA のみに特化) について検討する。図 3は Type 3 回路の概要である。表 8に Type 3 回路の基本コンポーネントを示す。

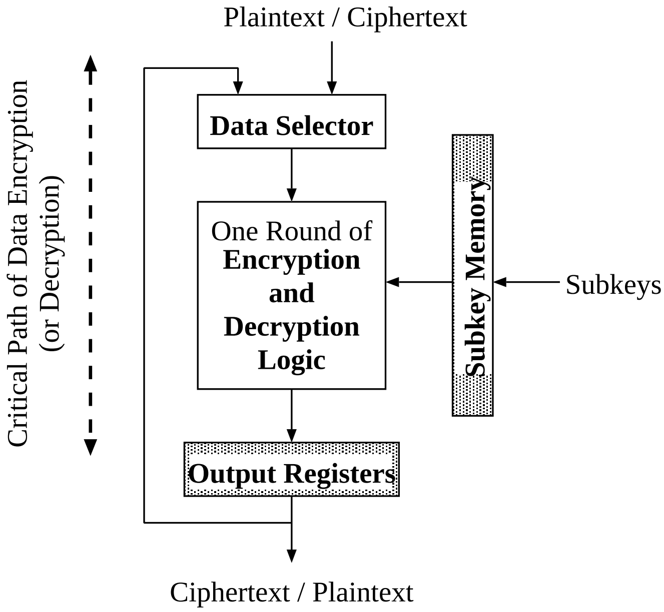


図 3: Type 3 回路の概要 (FPGA)

表 8: Type 3 回路の基本コンポーネント

暗号化及び 復号回路	組合せ回路から構成され、鍵拡大回路 (の一部) を含む 一段分の暗号化及び復号のためのデータ撹拌部回路
出力レジスタ	出力 (および中間処理状態の) データのためのレジスタ
データセレクタ	暗号化 (復号) データまたは出力データを選択するセレクタ
副鍵メモリ	外部からロードされた副鍵を蓄えるためのメモリ

これらの基本コンポーネントの設計方針を以下に示す。

1. “暗号化及び復号回路”

(a) ループアーキテクチャを採用する (一段分の回路をもつ)。

(b) パイプラインアーキテクチャは採用しない。

(c) 置換表は設計者の手によって最適化の後、論理合成ツールを用いて最適化を行う。

2. “出力レジスタ”、“副鍵メモリ” 及び “データセレクタ”

(a) 出力レジスタのサイズは 1 ブロック分 (=128 ビット) とする。

(b) 副鍵メモリのサイズはアルゴリズム内の全副鍵の長さとする。

(c) データセレクタは 1 ブロック分 (=128 ビット) の 2-1 セレクタとする。

以上の設計方針のもとで、 Camellia を FPGA に実装した場合について評価検討を行った。それらの評価結果は第 3章において表 3 に示したとおりである。ここで “スループット (Throughput)” は次の式で定義されるものとする:

$$\text{スループット [b/s]} = \frac{\text{ブロックサイズ (128 [bits])}}{\text{暗号化 (復号) 回路のクリティカルパス [sec]} \times \text{レイテンシ}}$$

5.4 Type 4: 高速実装 -2 (パイプラインアーキテクチャ)

Type 4 では、回路規模には制限を持たせず、暗号化及び復号処理においてできるだけ高速実装を実現するような場合 (ここでは FPGA のみ) について検討する。 (パイプラインアーキテクチャは CBC, CFB, OFB などの暗号利用モードの処理を実現することは困難。) 図 4は Type 4 回路の概要である。表 9に Type 4 回路の基本コンポーネントを示す。

表 9: Type 4 回路の基本コンポーネント

暗号化及び復号回路	組合せ回路より構成された暗号化及び復号のためのデータ撹拌部回路、出力及び中間処理状態のデータのためのレジスタ (1 〜 n)
鍵拡大回路	組合せ回路より構成された、暗号化鍵から副鍵を生成する回路
副鍵レジスタ	鍵拡大回路部の出力データのためのレジスタ

これらの基本コンポーネントの設計方針を以下に示す。

1. “暗号化及び復号回路” 及び “鍵拡大回路”

(a) ループアーキテクチャは採用しない。

(b) パイプラインアーキテクチャを採用する。

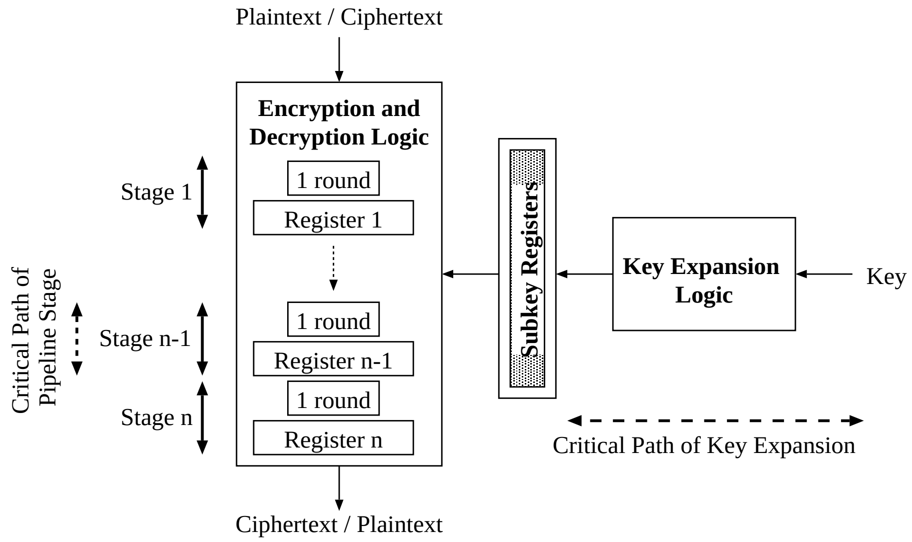


図 4: Type 4 回路の概要 (FPGA)

(c) 置換表は人手で最適化せず、論理合成ツールのみを用いて最適化を行う。

(d) ステージレジスタ (1 〜 n) の個々のサイズは、 1 ブロック分 (=128 ビット) とする。

2. “副鍵レジスタ”

(a) 副鍵レジスタのサイズはアルゴリズム内の全副鍵の長さ分とする。

以上の設計方針のもとで、 Camellia を FPGA に実装した場合について評価検討を行った。それらの評価結果は第 3章において表 3 に示したとおりである。ここで “スループット (Throughput)” は次の式で定義されるものとする:

$$\text{スループット [b/s]} = \frac{\text{ブロックサイズ (128 [bits])}}{\text{パイプラインステージのクリティカルパス [sec]}}$$

6　安全性

6.1　差分攻撃および線形攻撃に対する安全性

多くのブロック暗号に対する攻撃方法で最もよく知られておりかつ強力なものに Biham と Shamir が提案した差分攻撃 [BS93] と、松井が提案した線形攻撃 [M94] がある。これらの攻撃に対する安全性を評価する手法はいくつかあり、またそれらの間には一種の "双対性" という関係 [M95, CV95] が存在している。つまり、両者の攻撃に対する安全性は同様の方法によって評価することができる。

いくつかのブロック暗号では、何段かの段数における差分活性置換表や線形活性置換表の最少個数を使って、差分特性確率や線形特性確率の上界を評価できることが知られている。神田は、伝統的な 1 段換字置換網 (S 関数 -P 関数) によるラウンド関数を利用した Feistel 暗号の差分活性置換表と線形活性置換表の最少個数を示している [K00]。これ以降、線形変換関数 (P 関数) は全単射であると仮定する。

定義 1 P 関数での分岐数 $\mathcal{B}$ を、

$$\mathcal{B} = \min_{x \neq 0}(w_{\mathrm{H}}(x) + w_{\mathrm{H}}(P(x))),$$

と定義する。ここで $w_{\mathrm{H}}(x)$ は x のバイト単位のハミング重みを示す。

定義 2 差分活性置換表とは、非ゼロ入力差分が与えられた置換表のことであると定義する。また、線形活性置換表とは、非ゼロ出力マスク値が与えられた置換表のことであると定義する。

定理 1 任意の連続する 8 段における差分活性置換表・線形活性置換表の最少個数は、 $2\mathcal{B}+1$ 以上である。

定義 3 任意に与えられる Δx, Δy, Γx, $\Gamma y \in \mathrm{GF}(2^m)$ に対して、置換表 s_i: $\mathrm{GF}(2^m) \to \mathrm{GF}(2^m)$ の差分確率と線形確率を以下のように定義する。

$$\begin{aligned}
\Pr_x[s_i(x) \oplus s_i(x \oplus \Delta x) = \Delta y] &= \frac{\#\{x \in \mathrm{GF}(2^m) | s_i(x) \oplus s_i(x \oplus \Delta x) = \Delta y\}}{2^m} \\
\Pr_x[x \cdot \Gamma x = s_i(x) \cdot \Gamma y] &= \frac{\#\{x \in \mathrm{GF}(2^m) | x \cdot \Gamma x = s_i(x) \cdot \Gamma y\}}{2^m}
\end{aligned}$$

定義 4 p_s と q_s を全ての置換表 $\{s_1, s_2, \ldots\}$ での最大差分確率と最大線形確率であるとする。すなわち、

$$\begin{aligned}
p_s &= \max_i \max_{\Delta x \neq 0, \Delta y} \Pr_x[s_i(x) \oplus s_i(x \oplus \Delta x) = \Delta y] \\
q_s &= \max_i \max_{\Gamma y \neq 0, \Gamma x} \left(2 \Pr_x[x \cdot \Gamma x = s_i(x) \cdot \Gamma y] - 1\right)^2
\end{aligned}$$

定理 2 $\mathcal{D}$ と $\mathcal{L}$ をそれぞれ全体の差分活性置換表と線形活性置換表の最少個数であるとする。このとき、最大差分特性確率と最大線形特性確率はそれぞれ $p_s^{\mathcal{D}}$ と $q_s^{\mathcal{L}}$ で抑えられる。

Camellia は S 関数 -P 関数となるラウンド関数を利用している Feistel 暗号であるので、上述した技法により、最大差分特性確率と最大線形特性確率の上界を示すことによって、 Camellia がこれらの攻撃に対する耐性を有することを証明する。

Camellia の場合、置換表の最大差分確率と最大線形確率は

$$p_s = q_s = 2^{-6}$$

である。また、 P 関数の分岐数は 5、つまり

$$\mathcal{B} = 5$$

である。

$p,\ q$ をそれぞれ FL 関数と FL^{-1} 関数を除いた 16 段に削減した Camellia の最大差分特性確率と最大線形特性確率であるとすると、定理 1と定理 2から

$$p \le p_s^{2(2\mathcal{B}+1)} = (2^{-6})^{22} = 2^{-132}, \quad q \le q_s^{2(2\mathcal{B}+1)} = (2^{-6})^{22} = 2^{-132}$$

が得られる。どちらの確率も、 128 ビットブロック暗号の安全性閾値 2^{-128} を下回っている。このことから、 FL 関数と FL^{-1} 関数を除いた 15 段を超える段数 (16 段以上) に削減した Camellia には有効な差分特性または線形特性が存在しないことが導かれる。 FL 関数と FL^{-1} 関数は、任意の固定された鍵に対して、 (固定された) 線形変換であるので、それらが暗号の平均差分確率や最大線形確率をより大きくすることはない。それゆえ、 Camellia は差分攻撃と線形攻撃に対して十分な安全性を有することが証明される。

上記の結果は、定理 1と定理 2とに基づいていることに注意されたい。両方の定理とも 1 段換字置換網によるラウンド関数を利用した Feistel 暗号の一般的な場合を対象としているので、実際には、Camellia は上述した結果で示された安全性よりもさらに安全であると考えられる。そのことを補強する証拠として、我々は段数を削減した Camellia の活性置換表の個数を数えた。個数を数えるためのアルゴリズムは、以下の 3 点を除いて [M99] に記載されたアルゴリズムと同様である。

- 遷移確率表の代わりに活性置換表の個数を表した表を準備する
- 遷移確率を計算する代わりに活性置換表の個数を数える
- FL 関数と FL^{-1} 関数は、表中の全ての要素を活性置換表の最少個数に置き換える。このことは、 FL 関数と FL^{-1} 関数の前後の各々の差分特性や線形特性が高い確率、すなわち活性置換

表 10: Camellia の差分特性確率の上界

段数	1	2	3	4	5	6	7	8	9	10	11	12
定理 1と 2による			2^{-12}	2^{-30}		2^{-42}		2^{-66}				2^{-96}
評価			(2)	(5)		(7)		(11)				(16)
Camellia	1	2^{-6}	2^{-12}	2^{-42}	2^{-54}	2^{-66}	2^{-72}	2^{-72}	2^{-78}	2^{-108}	2^{-120}	2^{-132}
	(0)	(1)	(2)	(7)	(9)	(11)	(12)	(12)	(13)	(18)	(20)	(22)
FL/FL^{-1} 関数無	1	2^{-6}	2^{-12}	2^{-42}	2^{-54}	2^{-66}	2^{-78}	2^{-90}	2^{-108}	2^{-126}	2^{-132}	
	(0)	(1)	(2)	(7)	(9)	(11)	(13)	(15)	(18)	(21)	(22)	

注: () 内の数字は活性置換表の個数である

表 11: Camellia の線形特性確率の上界

# of rounds	1	2	3	4	5	6	7	8	9	10	11	12
定理 1と 2による			2^{-12}	2^{-30}		2^{-42}		2^{-66}				2^{-96}
評価			(2)	(5)		(7)		(11)				(16)
Camellia	1	2^{-6}	2^{-12}	2^{-36}	2^{-54}	2^{-66}	2^{-72}	2^{-72}	2^{-78}	2^{-102}	2^{-120}	2^{-132}
	(0)	(1)	(2)	(6)	(9)	(11)	(12)	(12)	(13)	(17)	(20)	(22)
FL/FL^{-1} 関数無	1	2^{-6}	2^{-12}	2^{-36}	2^{-54}	2^{-66}	2^{-78}	2^{-84}	2^{-108}	2^{-120}	2^{-132}	
	(0)	(1)	(2)	(6)	(9)	(11)	(13)	(14)	(18)	(20)	(22)	

注: () 内の数字は活性置換表の個数である

表の最少個数と等しくなるように連結する可能性が存在するので、これらの関数を挿入したことによる上記の弱鍵の存在を考慮していることを意味する。

結果として、 FL 関数と FL^{-1} 関数を含む 12 段の Camellia は 2^{-128} 以上となる確率を有する差分特性および線形特性が存在しないことを確認した。(表 10と表 11を参照のこと)

6.2 丸め差分攻撃

丸め差分 (truncated differentials) を利用した攻撃は Knudsen によって提案された [K95]。彼は、丸め差分を差分の一部分だけを予測できるような差分であると定義した。この丸め差分の概念は広いが、バイト単位の暗号ではバイト単位の差分を丸め差分として研究するのが自然である [MT99]。

マルコフ暗号では、差分 (differential) は同じ入力差分と同じ出力差分をもつ全ての差分特性の集

合であるので、最大差分確率は差分解読法に対する厳密な安全性評価を与えると考えられているが [LMM91]、その値を計算することは一般には不可能である。一方、丸め差分は、解読に利用することができる差分の部分集合とみなすことができる。例えば、バイト単位の暗号などいくつかの暗号では、丸め差分の確率は簡単かつ正確に計算することができ、最大差分特性確率よりも厳密な評価を与えることになる。

E2 の段数削減版に対する丸め差分攻撃は、松井と時田によって FSE'99 で発表された [MT99]。彼らの解読は、バイト内での差分の値を非ゼロとゼロとで区別する "バイト特性" を基にしている。結果として、 IT(初期変換) 関数と FT(最終変換) 関数を除いた 8 段の E2 に対して攻撃可能となるような 7 段バイト特性を発見した。文献 [MSAK00] に示すように、 E2 に対する最も優れた攻撃では、 IT 関数または FT 関数のどちらかを含んだ 8 段の E2 を 2^{94} 個の選択平文によって解読できる。さらに同じ文献 [MSAK00] で、 IT 関数と FT 関数を含んだ 7 段の E2 を 2^{91} 個の選択平文を使ってランダム置換と区別する攻撃も示されている。

Camellia は E2 と同様のバイト単位の暗号であり、丸め差分に対する安全性を評価することは重要である。文献 [MT99, MSAK00] に記述したのと同様のアルゴリズムを使って丸め差分を探索した。E2 と Camellia とのラウンド関数の大きな相違は、 2 段換字置換網 (つまり S-P-S) ではなく、 1 段換字置換網を採用したことである。 E2 の丸め差分の探索では、 Feistel 構造の排他的論理和でのバイト単位の差分消失確率として約 2^{-8} を利用した。しかし、 Camellia のラウンド関数では第 2 の S 関数がないため、複数バイトの差分消失確率が時々同時に約 2^{-8} になることがある。結果として、 Feistel 構造の排他的論理和での消失ルールを変更することによって、 FL 関数と FL^{-1} 関数の有無に関わらず、 10 段を超える (11 段以上の) 段数の Camellia はランダム置換と識別不可能である。

最近、杉田らによる Camellia(FL 関数と FL^{-1} 関数無し) の丸め差分攻撃と不能差分利用攻撃の結果が ASIACRYPT2001 に採録された [SKI01]。彼らは同じ入出力差分パターンをもつ 2 つの非自明な 9 段丸め差分を発見し、これを用いて初期 / 最終排他的論理和と FL 関数と FL^{-1} 関数を除いた 11 段 Camellia を攻撃するというものである。しかしながら、我々はこの丸め差分を用いて何段まで攻撃できるかはまだ未解決であると考えている。

6.3 丸め線形攻撃

我々は、丸め線形攻撃と呼ばれる新しい攻撃法を提案した。差分攻撃と線形攻撃の間の双対性により、上記と同様のアルゴリズムを使うことによって丸め線形攻撃に対する安全性を評価できる。具体的には、 P 関数の行列をその転置行列に置き換えることによって探索を実行することができる。結果として、 FL 関数と FL^{-1} 関数を除いた 10 段を超える (11 段以上の) 段数の Camellia はランダム置換と識別不可能である。

6.4 不能差分利用攻撃

不能差分とは、確率が 0 となる差分もしくは決して存在しない差分のことを意味する。そのような不能差分を使うと、副鍵の候補を絞ることが可能である。全単射のラウンド関数を利用するあらゆる Feistel 構造では少なくとも一つの 5 段不能差分が存在することが知られている。 Camellia は (6 段ごとに FL 関数と FL^{-1} 関数が挿入された)Feistel 構造であり、かつラウンド関数が全単射であるので、 Camellia には 5 段不能差分がある。また最新情報として、杉田ら [SKI01] が 7 段不能差分を発見している (但し、 FL 関数と FL^{-1} 関数を除いた場合)。 FL 関数と FL^{-1} 関数は鍵の値に応じて差分経路を変化させるため、これらの関数が不能差分を使って Camellia を攻撃することを困難にすると考えられる。したがって、仕様どおりの Camellia は不能差分を利用した攻撃法によって破られることはないと考えられる。

6.5 ブーメラン攻撃

ブーメラン攻撃は 2 つの差分を利用する [W99]。この 2 つの差分の成立確率を p_Δ, p_∇ とする。ブーメラン攻撃が全数探索より効率がよいためには

$$p_\Delta p_\nabla \geq 2^{-64} \tag{6}$$

が成り立つ必要がある。表 10から FL 関数と FL^{-1} 関数を除いた Camellia については、不等式 (6) を満たす組合せがないことがわかる。 FL 関数と FL^{-1} 関数を除いた Camellia に対する最適なブーメラン確率の上限は 8 段であり、 3 段 ($p_\Delta = 2^{-12}$) と 5 段 ($p_\nabla = 2^{-54}$) の差分の組合せで抑えられる。 FL 関数と FL^{-1} 関数を除いた Camellia の攻撃可能段数は Camellia の仕様である 18 段より極めて少ないので、 Camellia はブーメラン攻撃に対して安全であると考えられる。

6.6 高階差分攻撃

高階差分攻撃は、次数の低いブール多項式として表現可能な暗号に対して一般的に適用可能である。文献 [JK97, 定理 1] に記述されている高階差分攻撃では、中間ビットが少なくとも d 次のブール多項式によって表現されている場合、そのブール多項式の $d+1$ 次差分は 0 になる性質が利用されている。

置換表のブール多項式の次数 Camellia では置換表として GF(2^8) 上の逆数関数とのアフィン等価関数が採用されている。 GF(2^8) 上の逆数関数の各出力ビットのブール多項式の次数は 7 次であることが知られているが、 Camellia の置換表の次数は自明ではない (入出力にアフィン変換が追加されて

いるため)。そこで、我々は置換表の各出力ビットのブール多項式を見つけることによって、その次数が 7 次であることを確認した。

暗号全体のブール多項式の次数　Camellia では、暗号化処理中の中間ビットの次数は、データが置換表 (次数 7) を通過するにしたがって増加すると期待される。従って仕様どおりの Camellia に対して高階差分攻撃は成功しないと考えられる。しかしながら、高階差分攻撃には他にもいろいろなアプローチがあるため、 Camellia の高階差分攻撃についてはまだ検討の余地があると考えられる。

文献 [KK01] において川端らは、鍵長が 256 ビットの場合、 FL 関数と FL^{-1} 関数を除いた Camellia が 10 段まで、鍵の全数探索よりも速く攻撃が可能であることを示した。この攻撃は鍵長が 192 ビットの場合 9 段まで、鍵長が 128 ビットの場合 8 段まで適用可能である。この攻撃は「高階差分攻撃」と名付けられていたが、用いられているアプローチはスクエア攻撃と同様である。

6.7　スクエア攻撃

スクエア攻撃は Square 暗号 [DKR97] のバイト構造を利用した Square 暗号用の攻撃として提案された。この攻撃は Rijndael, Hierocrypt, Camellia のようなバイト構造をもつ暗号に適用できる。我々の行なった Camellia に対するスクエア攻撃に対する評価は 6.8章を参照のこと。

スクエア攻撃のアプローチは高階差分攻撃と似ている。すなわち、攻撃者は平文のある集合をあつめて、暗号の何段後かに確率 1 で成り立つ、鍵に依存しない性質を予測するというアプローチである。川端らによる高階差分攻撃 [KK01] もこのアプローチをとっている。

He と Qing による 6 段の Camellia に対するスクエア攻撃についての論文が ICICS 2001 に採録されている [HQ01]。この攻撃では 6 段の Camellia に対する攻撃の計算量は川端らの攻撃より多い (攻撃 [HQ01] では 2^{112} 回の暗号化が必要なのに対し、攻撃 [KK01] では $2^{22}/6$ 回の暗号化が必要) が、必要な平文数は少ない (攻撃 [HQ01] では 13×2^8 個の平文が必要なのに対し、攻撃 [KK01] では 2^{17} 個の平文が必要)。

6.8　補間攻撃と線形和攻撃

文献 [JK97] で提案された補間攻撃は単純な代数的な関数を利用している暗号に対する典型的な攻撃である。

補間攻撃の原理は、大雑把に言うと、もし未知係数の個数が N である平文の多項式または有理表現として暗号文が表現されるならば、 N 個の平文と暗号文の組を使って、その多項式または有理表現を構成することができる。一度、攻撃者がその多項式または有理表現を構成すると、攻撃者は鍵を知ることになしに、任意の平文をその鍵に対応する暗号文へ暗号化し、また任意の暗号文を対応する

平文へ復号することが可能である。 N は攻撃複雑度と攻撃に必要な組数を決めるので、 N をできるだけ大きくすることが重要である。もし、攻撃者が N 個の平文と暗号文の組を集めることが現実的でないほど N が大きければ、その暗号は補間攻撃に対して安全である。

線形和攻撃 [A00] は補間攻撃 [JK97] の一般化である。線形和攻撃に対する実用的な安全性評価アルゴリズムが文献 [A00] で提案されている。このアルゴリズムを用い、任意の平文バイトと暗号文バイトの GF(2^8) 上の線形和関係を探索した。表 12にその結果をまとめた。

表 12: 128, 192, 256 ビット鍵に対する最小未知係数の数

初期排他的論理和 + r 段 ($r < 4$)	1
初期排他的論理和 + 4 段	255
それ以上の段数	256

表 12によると、 Camellia は補間攻撃を含む線形和攻撃に対して安全なことがわかる。このことは、文献 [A00, Theorem 3] によると、 Camellia が Square 攻撃 [DKR97] に対しても安全であることを意味している。

6.9 等価鍵不存在性

鍵スケジュールによって生成される副鍵集合には元の秘密鍵を含んでいるので、別の秘密鍵から生成される副鍵集合に等価な集合は存在しない。それゆえ、異なる秘密鍵が多くの平文の各々に対して同じ暗号文に暗号化することになるようなことは存在しないと考えられる。

6.10 スライド攻撃

先駆的研究 [B94, K93] を基にしたスライド攻撃が提案されている [BW99, BW00]。特に、同一のラウンド関数、すなわち全てのラウンド関数が同じ構造かつ同じ副鍵を利用しているような繰り返し暗号は、スライド攻撃に弱点を有することが示された。

Camellia では、ラウンド間の規則性を崩すために Feistel 構造の 6 段ごとに FL 関数と FL^{-1} 関数が挿入されている。さらに鍵スケジュールの観点から、スライド攻撃は非常に成功しにくいと考えられる。 (第 6.11章を参照のこと)

6.11 関連鍵攻撃

Camellia の鍵スケジュールは関連鍵攻撃 [B94, KSW96] を非常に難しくすると考えられる。これらの攻撃では、攻撃者はいくつかの関連鍵を使って暗号化を行うことができなければならない。いってみれば、二つの鍵の間の関係を知っているとすると、副鍵の対応する関係を予測できさえすれば、それらの鍵が異なる平文の組をどのように暗号化するかを予測することが可能になる。しかし、副鍵は秘密鍵及びその暗号化の結果である K_A と K_B に依存しているので、攻撃者が望んだ副鍵の関係を持つような秘密鍵を得たいとしてもそのような秘密鍵を得ることができない。また逆に、秘密鍵を変えることによって副鍵の関係をコントロールしたり予測することは非常に難しい。

6.12 統計量情報による評価

ほとんどの統計的特性は差分攻撃やその他の暗号学的な攻撃に依存している。例えば平文の 1 ビットを反転させたとき、暗号文の何ビットが反転するかということがよく議論される。差分分布表の定義によれば、差分攻撃に対する耐性から反転ビット数がほぼ半分であることが導ける。もちろん、十分な計算機資源があれば、統計的な弱点を発見できるかもしれない。しかしながら 128 ビットブロック暗号に対してそのような統計的特性を計算できるほどの計算機資源は構築されていない。

また、次のことに注意されたい。計算機資源の制約から、しばしばラウンド関数に対して統計的性質が調べられる。しかしながら、我々はこのような探索はほとんど意味がないと考える。何故ならば、ラウンド関数に対して統計的に良い性質を満たすが、全体では統計的に良くない性質が存在する暗号を構成することも、また逆に、ラウンド関数に対して統計的に良くない性質があるが、全体では統計的に良い性質を満たす暗号も構成することも可能であるからである。

暗号技術評価報告書 (CRYPTREC Report 2000)[C01] において、アバランシュ性評価が行なわれ、 Camellia については、ラウンド関数では期待値から離れている部分があるが、データランダム化部では 4 段以降の撹拌に特徴は見られないことが報告されている。

6.13 実装攻撃 (サイドチャネル攻撃)

時間攻撃 (timing attacks)[K96] や、電力解析攻撃 (power analysis attacks)[KJJ99] によって、十分な安全対策が取られていない実装から秘密情報を取り出すことが可能であることが知られている。文献 [DR99] で提案されている分類法を用いると、 Camellia は論理命令と表参照と固定ビット循環シフトしか用いてないので、**有利** (favorable) に分類される。

一方、 Chari らは、全ての AES 候補暗号は電力解析攻撃に対して安全であるかどうか疑わしいと主張している [CJRR99]。これら 2 つの論文が矛盾しているように電力差分攻撃の研究ははじまったばかりなので、この攻撃に対する安全性は不明である。我々は現在の技術を考えると、 Camellia は

ハードウェア技術によって守られるべきであり、仕様からそのまま導ける安全性によっては評価すべきではないと考える。近い将来に電力差分攻撃の研究が進展することを期待する。

6.14 ブルートフォース攻撃

多くのブルートフォース攻撃は任意のブロック暗号に適用可能であり、攻撃複雑度は、攻撃対象の暗号の構造にかかわらず、ブロックサイズと鍵サイズにだけ依存している* Camellia は 128 ビットのブロックサイズであり、 128 ビット、 192 ビット、 256 ビットの 3 つの鍵サイズを認めている。以下の議論では、 k は鍵のビットサイズを表す。

鍵の全数探索： 鍵の全数探索では、攻撃者は一組の平文と ECB モードで暗号化した暗号文を得ることができるならば、攻撃者は可能な 2^k 個全ての鍵について平文を暗号化することによって正しい鍵を発見できる。

暗号の鍵スケジュールの弱点が鍵の全数探索攻撃の効率を改善することがあるが [K94]、 Camellia ではそのような弱点を我々は発見できなかった。鍵の全数探索の攻撃複雑度は平均約 2^{k-1} 回の暗号化であると評価されている。したがって、必要な攻撃複雑度は、 128 ビット、 192 ビット、 256 ビットの鍵それぞれに対して 2^{127}、 2^{191}、 2^{255} の暗号化時間相当となる。それゆえ、鍵の全数探索に対する Camellia の安全性は十分である。

時間メモリトレードオフ攻撃： 平文の中でしばしば使われるいくつかの単語がある。もし、攻撃者がそのような単語で作られる平文ブロックを 2^k 個の鍵を使って暗号化し、それらを 2^k 個の暗号文に相当するメモリ空間に蓄えていたとすると、攻撃者は、対応する暗号文を得た後、対応する鍵を見つけるために該当する暗号文を探し出すだけでよくなる。この攻撃は表攻撃と呼ばれる。この攻撃では、2^k 回の暗号化を終えた後、攻撃時間複雑度が鍵の全数探索の場合よりもはるかに小さくなる。

時間メモリトレードオフ攻撃 [H80, KM96] では、鍵の全数探索で必要とする攻撃時間複雑度と表攻撃で必要とする空間複雑度の両方を劇的に減少させることができる。

しかしながら、上記の 2 つの攻撃は、ともに鍵の全数探索の攻撃時間複雑度に等しいだけの事前計算を必要とする。 Camellia が認める鍵サイズは、今日の技術による鍵の全数探索に対して安全であるのに十分な長さである。

辞書攻撃： 辞書攻撃では、攻撃者は同じ鍵を利用した平文と暗号文の組を集め、それらを “辞書” として置いておく。攻撃者は、その鍵で暗号化された暗号文を見つけたときだけ、辞書の中にあるか

*厳密にいえば、攻撃に必要な計算時間はそのブロック暗号の性能に依存する。しかし、性能は暗号化時間にだけ影響し、攻撃時間複雑度では無視できる量しか変化しない。

を調べる。もし辞書の中にあれば、攻撃者はすでに平文を所有していることになる。Camellia のブロックサイズは 128 ビットであるので、辞書攻撃では、攻撃者が未知の鍵で任意のメッセージを暗号化または復号することを可能にする 2^{128} 個の異なる平文ブロックを蓄積するだけの空間を必要とする。成功確率は辞書空間に依存し、ブロックサイズが大きくなるにつれて、同じ成功確率を達成するのに必要となる空間は指数関数的に増大する。128 ビットブロック暗号である Camellia は、この攻撃に対して十分安全である。

暗号文一致攻撃： 暗号文一致攻撃では、ECB モード、CBC モード、CFB モードのようないくつかのモード操作に対して、全ての暗号文のおおよそ平方根の個数が利用できるとき、“バースデーパラドックス” によっておよそ $\frac{1}{2}$ 以上の確率で、同一の暗号文ブロックが存在することが期待できる。そのとき、平文に関する価値のある情報を得ることができる。この攻撃がブロックサイズに依存することに注意されたい。Camellia のブロックサイズは 128 ビットであるので、同じ鍵で 2^{64} 個と同じぐらい多くのブロックの暗号化を実行することがないならば、この攻撃に対する脅威は小さい。

7　まとめ

本稿では、 Camellia とその設計指針、ソフトウェアとハードウェア実装に対する適合性、ならびに我々の解析結果を示した。

本稿で示されている性能にはさらなる最適化の余地が残されている。最新の性能結果は、 Camellia ホームページ： `http://info.isl.ntt.co.jp/camellia/` 上にて更新される。

我々が Camellia を解析した結果、重大な弱点は発見されなかった。この暗号は従来からある設計方針を踏襲しており、 Camellia に対する現実的な攻撃が発見されるためには暗号解読の分野における非常に大きな突破口が必要になるだろう。 Camellia が現存する最良のブロック暗号の安全性と同等の、非常に安全な暗号であると我々は考えている。

参考文献

[A00] K. Aoki. Practical Evaluation of Security against Generalized Interpolation Attack. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*, Vol. E83-A, No. 1, pp. 33–38, 2000. (A preliminary version was presented at SAC'99).

[ABK98] R. Anderson, E. Biham, and L. Knudsen. Serpent: A Flexible Block Cipher With Maximum Assurance. In *The First AES Candidate Conference*, 1998.

[AIK+00a] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. Implementations of the 128-bit block cipher – *Camellia* –. Technical Report ISEC2000-73, The Institute of Electronics, Information and Communication Engineers, 2000. (in Japanese).

[AIK+00b] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. *Camellia*: A 128-Bit Block Cipher Suitable for Multiple Platforms – Extended Abstract –. In *First NESSIE Workshop*, 2000.

[AU00] K. Aoki and H. Ueda. Optimized Software Implementations of E2. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*, Vol. E83-A, No. 1, pp. 101–105, 2000. (The full paper is available on `http://info.isl.ntt.co.% linebreak[3]jp/e2/RelDocs/`).

[B94] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. *Journal of Cryptology*, Vol. 7, No. 4, pp. 229–246, 1994. (The extended abstract was appeared at EUROCRYPT'93).

[BS93] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard.* Springer-Verlag, Berlin, Heidelberg, New York, 1993.

[BW99] A. Biryukov and D. Wagner. Slide Attacks. In L. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 245–259, Berlin, Heidelberg, New York, 1999. Springer-Verlag.

[BW00] A. Biryukov and D. Wagner. Advanced Slide Attacks. In S. Vaudenay, editor, *Advances in Cryptology — EUROCRYPT2000*, Volume 1807 of *Lecture Notes in*

Computer Science, pp. 589–606, Berlin, Heidelberg, New York, 2000. Springer-Verlag.

[C98] Compaq Computer Corporation. *Alpha Architecture Handbook (Version 4)*, 1998. (You can download the manual from Compaq's technical documentation library: `http://www.support.compaq.com/alpha-tools/documentation/current/chip-docs.html`).

[C01] CRYPTREC. CRYPTREC Report 2000, April 2001.

[CJRR99] S. Chari, C. Jutla, J. R. Rao, and P. Rohatgi. A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. In *Second Advanced Encryption Standard Candidate Conference*, pp. 133–147, Hotel Quirinale, Rome, Italy, 1999. Information Technology Laboratory, National Institute of Standards and Technology.

[CV95] F. Chabaud and S. Vaudenay. Links Between Differential and Linear Cryptanalysis. In A. D. Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, Volume 950 of *Lecture Notes in Computer Science*, pp. 356–365. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[DKR97] J. Daemen, L. R. Knudsen, and V. Rijmen. The Block Cipher SQUARE. In E. Biham, editor, *Fast Software Encryption — 4th International Workshop, FSE'97*, Volume 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Berlin, Heidelberg, New York, 1997. Springer-Verlag.

[DR98] J. Daemen and V. Rijmen. *AES Proposal: Rijndael*, 1998. (`http://www.esat.kuleuven.ac.be/~rijmen/rijndael/`).

[DR99] J. Daemen and V. Rijmen. Resistance Against Implementation Attacks. A Comparative Study of the AES Proposals. In *The Second AES Candidate Conference*, 1999.

[F00] A. Fog. *How to optimize for the Pentium microprocessors*, 2000. (`http://www.agner.org/assem/`).

[H80] M. Hellman. A Cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, Vol. IT-26, No. 4, pp. 401–406, 1980.

[HQ01] Y. He and S. Qing. Square Attack on Reduced Camellia Cipher. submitted to the 3rd International Conference on Information and Communications Security (ICICS 2001), 2001.

[I99] Intel Corporation. *Intel Architecture Software Developer's Manual (Volume 2: Instruction Set Reference)*, 1999. (You can download the manual from Intel's developer site: `http://developer.intel.com/`).

[ISKM01] T. Ichikawa, T. Sorimachi, T. Kasuya, and M. Matsui. On the criteria of hardware evaluation of block ciphers (1). Technical Report ISEC2001-53, The Institute of Electronics, Information and Communication Engineers, 2001. (in Japanese).

[JK97] T. Jakobsen and L. R. Knudsen. The Interpolation Attack on Block Cipher. In E. Biham, editor, *Fast Software Encryption — 4th International Workshop, FSE'97*, Volume 1267 of *Lecture Notes in Computer Science*, pp. 28–40, Berlin, Heidelberg, New York, 1997. Springer-Verlag.

[K93] L. R. Knudsen. Cryptanalysis of LOKI91. In J. Seberry and Y. Zheng, editors, *Advances in Cryptology — AUSCRYPT'92*, Volume 718 of *Lecture Notes in Computer Science*, pp. 196–208. Springer-Verlag, Berlin, Heidelberg, New York, 1993.

[K94] L. R. Knudsen. Practically secure Feistel ciphers. In R. Anderson, editor, *Fast Software Encryption 1993 — Cambridge Security Workshop (FSE1)*, Volume 809 of *Lecture Notes in Computer Science*, pp. 211–221, Berlin, Heidelberg, New York, 1994. Springer-Verlag.

[K95] L. R. Knudsen. Truncated and Higher Order Differentials. In B. Preneel, editor, *Fast Software Encryption — Second International Workshop*, Volume 1008 of *Lecture Notes in Computer Science*, pp. 196–211. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[K96] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO'96*, Volume 1109 of *Lecture Notes in Computer Science*, pp. 104–113. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[K00] M. Kanda. Practical Security Evaluation against Differential and Linear Attacks for Feistel Ciphers with SPN Round Function. In *SAC2000, Seventh Annual Workshop on Selected Areas in Cryptography, 14-15 August 2000, Workshop Record*, 2000.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology — CRYPTO'99*, Volume 1666 of *Lecture Notes in Computer Science*, pp. 388–397. Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[KK01] T. Kawabata and T. Kaneko. A Study on Higher Order Differential Attack of Camellia. In *Second NESSIE Workshop*, 2001. (This paper is based on T. Kawabata, Y. Ohgaki and T. Kaneko, "A Study on Strength of Camellia against Higher Order Differential Attack," (in Japanese), Technical report of IEICE, ISEC2001-9, pp.55–62, The Institute of Electronics, Information and Communication Engineers, 2001.).

[KM96] K. Kusuda and T. Matsumoto. Optimization of Time-Memory Trade-Off Cryptanalysis and Its Application to DES, FEAL-32, and Skipjack. *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan)*, Vol. E79-A, No. 1, pp. 35–48, 1996.

[KMA+98] M. Kanda, S. Moriai, K. Aoki, H. Ueda, M. Ohkubo, Y. Takashima, K. Ohta, and T. Matsumoto. A New 128-bit Block Cipher ***E2***. Technical Report ISEC98-12, The Institute of Electronics, Information and Communication Engineers, 1998. (in Japanese).

[KSW96] J. Kelsey, B. Schneier, and D. Wagner. Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO'96*, Volume 1109 of *Lecture Notes in Computer Science*, pp. 237–251. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[LMM91] X. Lai, J. L. Massey, and S. Murphy. Markov Ciphers and Differential Cryptanalysis. In D. W. Davies, editor, *Advances in Cryptology — EUROCRYPT'91*, Volume 547 of *Lecture Notes in Computer Science*, pp. 17–38. Springer-Verlag, Berlin, Heidelberg, New York, 1991.

[M94] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In T. Helleseth, editor, *Advances in Cryptology — EUROCRYPT'93*, Volume 765 of *Lecture Notes in Com-*

puter Science, pp. 386–397. Springer-Verlag, Berlin, Heidelberg, New York, 1994. (A preliminary version written in Japanese was presented at SCIS93-3C).

[M95] M. Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In A. D. Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, Volume 950 of *Lecture Notes in Computer Science*, pp. 366–375. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[M97] M. Matsui. New Block Encryption Algorithm MISTY. In E. Biham, editor, *Fast Software Encryption — 4th International Workshop, FSE'97*, Volume 1267 of *Lecture Notes in Computer Science*, pp. 54–68, Berlin, Heidelberg, New York, 1997. Springer-Verlag. (A preliminary version written in Japanese was presented at ISEC96-11).

[M99] M. Matsui. Differential Path Search of the Block Cipher E2. Technical Report ISEC99-19, The Institute of Electronics, Information and Communication Engineers, 1999. (in Japanese).

[MIYY88] M. Matsui, T. Inoue, A. Yamagishi, and H. Yoshida. A note on calculation circuits over GF(2^{2n}). Technical Report IT88-14, The Institute of Electronics, Information and Communication Engineers, 1988. (in Japanese).

[MSAK00] S. Moriai, M. Sugita, K. Aoki, and M. Kanda. Security of E2 against Truncated Differential Cryptanalysis. In H. Heys and C. Adams, editors, *Selected Areas in Cryptography — 6th Annual International Workshop, SAC'99*, Volume 1758 of *Lecture Notes in Computer Science*, pp. 106–117, Berlin, Heidelberg, New York, 2000. Springer-Verlag.

[MT99] M. Matsui and T. Tokita. Cryptanalysis of a Reduced Version of the Block Cipher E2. In L. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 71–80, Berlin, Heidelberg, New York, 1999. Springer-Verlag. (Japanese version was presented at SCIS99.).

[RDP+96] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win. The Cipher SHARK. In D. Gollmann, editor, *Fast Software Encryption — Third Interna-*

tional Workshop, Volume 1039 of *Lecture Notes in Computer Science*, pp. 99–111. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[SKI01] M. Sugita, K. Kobara, and H. Imai. Security of Reduced Version of the Block Cipher Camellia against Truncated and Impossible Differential Cryptanalysis. submitted to ASIACRYPT 2001, 2001.

[W99] D. Wagner. The Boomerang Attack. In L. R. Knudsen, editor, *Fast Software Encryption — 6th International Workshop, FSE'99*, Volume 1636 of *Lecture Notes in Computer Science*, pp. 156–170, Berlin, Heidelberg, New York, 1999. Springer-Verlag.

[Y01a] C.-H. Yang. Performance Evaluation of AES/DES/Camellia on the 6805 and H8/300 CPUs. In *Proceedings of the 2001 Symposium on Cryptography and Information Security*, Volume II of *SCIS2001*, pp. 727–730, Oiso, Japan, 2001. Technical Group on Information Security (IEICE).

[Y01b] C.-H. Yang. Supplementary information for C.H. Yang SCIS 2001 paper. http://www.geocities.com/chyang00/SCIS2001, 2001.

A 更新履歴

第 1.1 版 (2001 年 8 月 28 日)

- 3 章 表 1 「Camellia のソフトウェア実装の性能」中の罫線を適切に修正 (現在はこの表はありません)
- 6 章 表 10「Camellia の差分特性確率の上界」の FL/FL^{-1} 関数無の場合の 4 段 Camellia の評価結果の誤植を修正

第 2.0 版 (2001 年 9 月 26 日)

- 要旨 性能数値例を最新の情報に更新。
- 1 章 「将来の標準化活動」の段落を現状にあわせて改め、タイトルも「標準化活動」と変更。
- 3 章 実装評価を最新の情報に更新。
- 4 章 4.2.7 章 式 (3) 中の SP_1, SP_2, SP_3, SP_4 のみを用いて実装する場合の計算式の誤植を訂正。
- 5 章 ハードウェア実装評価に最新の情報を追加。
- 6.2 章 (丸め差分攻撃) に最新解析結果を追加。
- 6.4 章 (不能差分利用攻撃) に最新解析結果をもとに更新。
- 6.6 章 (高階差分攻撃) 最新解析結果をもとに更新。
- 6.7 章 (スクエア攻撃) を追加。
- 6.12 章 (統計量情報による評価) に情報を追加。